UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

ExperimentaUnirioja. Aplicación web para la gestión de talleres de la Facultad de Ciencia y Tecnología

Autor/es

RAÚL ADÁN SAN MARTÍN

Director/es

ELOY JAVIER MATA SOTÉS

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2018-19

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Titulación

ExperimentaUnirioja. Aplicación web para la gestión de talleres de la Facultad de Ciencia y Tecnología

Realizado por:

Raúl Adán San Martín

Tutelado por:

Eloy Javier Mata Sotés

Logroño, Junio, 2019

Resumen:

Experimenta Unirioja es un conjunto de talleres de Ingeniería Informática, Matemáticas, Química e Ingeniería Agrícola organizados por la universidad y orientados a alumnos de cuarto de la ESO. El desarrollo de este proyecto se ha llevado a cabo para permitir la automatización del proceso de inscripción a los talleres de las distintas actividades.

Este proyecto permitirá que los profesores de los centros puedan realizar las reservas para los talleres, editarlas e incluso eliminarlas, y que los organizadores de las actividades puedan gestionarlas de forma sencilla y cómoda. Además, la aplicación estará diseñada para evitar el mantenimiento de una persona técnica de forma anual.

Abstract:

Experimenta Unirioja is a set of workshops in Computer Engineering, Mathematics, Chemistry and Agricultural Engineering organized by La Rioja University and designed for “ESO fourth-year” students. The development of this project has been carried out to allow the automation of the registration process to the different activities.

Experimenta Unirioja will allow high school school teachers not only to manage the workshops reservations, but also to edit them and even to eliminate them. In addition, the organizers will be able to supervise the different activities in a simple and easy-to-use way. Furthermore, the web application will be configurated to avoid the annual maintenance of a technician.

CONTENIDO

Capítulo 1. Documento de Objetivos del Proyecto (DOP) 5

1. Introducción 5
2. Alcance: 5
3. Entregables: 6
4. Recursos humanos: 7
5. Plan de comunicaciones: 7
6. Tecnologías utilizadas: 7
7. Metodologías ágiles: 8
8. EDT: 9
9. Planificación: 10
10. Plan de dedicaciones: 11
11. Plan de contingencias: 12

Capítulo 2. Análisis 13

1. Definición de roles. 13
2. Definición de requisitos funcionales. 14
3. Definición de requisitos no funcionales. 17
4. Plan de pruebas. 17

Capítulo 3. Diseño: 19

1. Arquitectura 19
2. Diagrama de clases. 19
3. Diseño de la base de datos: 20
4. Interfaces. 23
5. Diseño de la API 25

Capítulo 4. Implementación 27

1. Construcción del front-end. 27
   - 1.1.1 Módulos: 27
   - 1.1.2 Componentes: 28
   - 1.1.3 Servicios: 32
   - 1.1.4 Construcción del calendario: 34
2. Construcción del back-end: 36
3. Construcción de la base de datos: 43
4. Librerías usadas: 44

Capítulo 5. Test y Pruebas 45

5.1 Pruebas 45

5.2 Test 46
Capítulo 6. Gestión del proyecto. 47
1. Alcance 47
1.1 Requisitos: 47
1.2 Incidencias y cambios: 47
2. Tiempo 48
2.1 Cronograma real: 48
2.2 Desviaciones: 49
2.3 Estimaciones de dedicación: 50
3. Comunicaciones 50
Capítulo 7. Conclusiones 51
1. Aprendizaje personal: 51
2. Mejoras: 51
Bibliografía: 53

CAPÍTULO 1. DOCUMENTO DE OBJETIVOS DEL PROYECTO (DOP)

1. INTRODUCCIÓN

a) Contexto:

Este trabajo de fin de Grado consistirá en realizar una aplicación web con la finalidad de gestionar de forma cómoda y automática las inscripciones a los talleres **Experimenta Unirioja** que organiza la Facultad de Ciencia y Tecnología de la Universidad de la Rioja.

Este proyecto permitirá facilitar el tratamiento de las reservas que se realizan en los talleres ofertados por la universidad. Se realizará una aplicación web que permita la automatización del proceso para evitar un complejo mantenimiento que requiera ayuda de terceras personas técnicas cada año.

Además, la aplicación proporcionará nuevas funcionalidades, tanto a los profesores que organicen talleres como a los usuarios que hagan reservas de plazas. El personal docente podrá controlar de forma directa la configuración de las actividades, así como tener una visión general de la evolución de las reservas. Los usuarios podrán reservar plazas para las actividades ofertadas y podrán gestionar posteriormente sus reservas registradas en el sistema.

b) Antecedentes:

Actualmente, el sistema se basa en una sencilla aplicación que permite realizar las subscripciones a los cursillos mediante el envío de un formulario. Los datos se guardan en una base de datos MySQL mal diseñada.

Cada año, es necesario realizar la configuración de la aplicación ya que no está automatizada ni permite que el personal docente controle de forma personal la gestión de los talleres. Por lo tanto, se necesita una persona que tenga conocimientos informáticos para reprogramarla y ajustarla cada año. En este proyecto, se intentará automatizar al máximo posible todos los procesos de configuración y además se añadirán nuevas funcionalidades que permitan a los responsables de los talleres tener una mayor libertad en el uso de la aplicación.

2. ALCANCE:

El objetivo principal de este proyecto es facilitar y automatizar lo máximo posible el proceso de inscripciones en los diferentes talleres que se oferten, así como la gestión de las reservas por el personal docente de la universidad.

Además, se realizará un aumento de la funcionalidad de los distintos tipos de usuarios de la aplicación. El personal docente podrá crear, eliminar y modificar los talleres que deseen impartir sin tener que necesitar una persona intermedia que se encargue de ello. Habrá también una visión global para ver las estadísticas de las reservas y las plazas disponibles para los talleres.

La aplicación también permitirá que representantes de centros educativos puedan realizar las inscripciones en las distintas actividades, así como que puedan visualizar las reservas que previamente hayan realizado.

3. ENTREGABLES:

En esta tabla, podemos ver los entregables planificados para el proyecto. Cada uno de ellos tiene un código que identifica la fase del proyecto en la que se realiza y otro número que nos indica el orden dentro de su fase.

Entregables del proyecto	
Código: E10-1	Título: Alcance del proyecto.
Contenido: definición de las metas y principales objetivos del proyecto.	
Código: E10-2	Título: Planificación del proyecto.
Contenido: planificación de horarios, plazos, hitos, tecnologías a usar, métodos de comunicación, recursos humanos y herramientas a utilizar.	
Código: E20-1	Título: Definición de roles
Contenido: definición de todos los tipos de usuarios de la aplicación, así como las funcionalidades de cada uno.	
Código: E20-3	Título: Requisitos funcionales
Contenido: definición de todas las funcionalidades que tiene que tener el software final.	
Código: E20-3	Título: Requisitos no funcionales
Contenido: definición de los requisitos no funcionales.	
Código: E20-4	Título: Definición de entidades.
Contenido: definición de las clases.	
Código: E20-5	Título: Plan de pruebas.
Contenido: realización de un plan para comprobar el correcto funcionamiento de cada funcionalidad tras su implementación.	
Código: E30-1	Título: Diseño de la arquitectura.
Contenido: elaboración del diseño de la arquitectura que tendrá el sistema.	
Código: E30-2	Título: Diseño de interfaces.
Contenido: elaboración de mockups para repartir las funcionalidades en distintas vistas.	
Código: E30-3	Título: Diseño de la base de datos.
Contenido: diseño de una base de datos no relacional.	
Código: E40-1	Título: Despliegue del servidor.
Contenido: montaje de un servidor y su configuración para hacerlo funcionar como servidor de aplicaciones.	
Código: E40-2	Título: Aplicación web
Contenido: desarrollo de la aplicación web.	
Código: E50-1	Título: Pruebas
Contenido: se realizará un apartado en el que se explicará cómo se ha comprobado cada funcionalidad de la aplicación.	
Código: E60-1	Título: Presentación
Contenido: presentación Power Point para la defensa del proyecto.	
Código: E70-1	Título: Documento de control y seguimiento
Contenido: apartado en el que se refleja la evolución del proyecto, el cumplimiento de los tiempos, hitos, desvíos y sus justificaciones. Así como la realización de planes de contingencia en caso de que los hubiera.	

Tabla 1

4. RECURSOS HUMANOS:

Las personas implicadas en este proyecto son:

- **Eloy Javier Mata**: como tutor del trabajo de fin de grado. Realizará las funciones de corrección de la documentación del proyecto, así como el guiado de las distintas fases del mismo.
- **Raúl Adán San Martin**: como responsable del proyecto. Se encargará de realizar la documentación, desarrollo e implementación del proyecto.

5. PLAN DE COMUNICACIONES:

Debido a mis circunstancias laborales, no dispongo de mucho tiempo para estar por la universidad, ni en Logroño con lo que las comunicaciones se harán de las siguientes formas:

- **Email**: La comunicación principal se hará mediante el correo electrónico para solucionar las dudas y para enviar toda la documentación del proyecto.
- **Reuniones**: debido a que no me encuentro actualmente en Logroño, se harán reuniones puntuales en los momentos en los que me encuentre en la ciudad con el propósito de verificar el avance correcto y enfocar debidamente las direcciones a seguir. Se harán actas de cada reunión que podrán verse en el anexo II del trabajo.
- **Teléfono**: se utilizará el teléfono para una comunicación más rápida e informal que permita resolver pequeñas dudas y concertar las reuniones.

6. TECNOLOGÍAS UTILIZADAS:

En este proyecto se usarán las siguientes tecnologías:

- **Angular 6.** Framework de desarrollo de aplicaciones web con el que se desarrollará todo el frontal de la aplicación.
- **NodeJS**. Entorno de ejecución JavaScript para el desarrollo de la API REST que conectará con la base de datos de la aplicación.
- **TypeScript** como lenguaje de programación en el framework angular.
- **JavaScript** como lenguaje de programación en el entorno NodeJS.
- **CSS**. Para aplicar estilos a la estructura de la web.
- **HTML**. Para la estructura de la página web.
- **MongoDB**. Base de datos NoSQL orientada a documentos de código abierto en la que se almacenarán todos los datos de la aplicación.

Las herramientas que se usarán durante el desarrollo del proyecto son:

- **Visual Studio Code**. Entorno de desarrollo orientado a aplicaciones web Angular.
- **MongoDB Compass**. Herramienta para gestionar todo lo relacionado con las bases de datos.
- **Github**. Repositorio para realizar un control de versiones.
- **MobaXterm**. Conexión para la transferencia de archivos mediante FTP.
- **Postman**. Programa para hacer pruebas contra los servicios REST.

7. METODOLOGÍAS ÁGILES:

Durante este proyecto utilizaré la metodología llamada **desarrollo en cascada**. Esta tecnología se basa en un conjunto de etapas que son ejecutadas secuencialmente. Cada vez que se termina una fase, se hace una revisión, si se ha terminado, se avanza a la siguiente fase y en caso contrario se repite la etapa.

Las principales ventajas del desarrollo en cascada son:

- Comenzar a desarrollar software con bastante rapidez.
- Estimar fechas con mayor precisión.
- Obtener niveles de satisfacción con el cliente más elevados.

He decidido utilizar esta metodología debido a que conozco bastante bien el proyecto y sé que tiene unos requisitos muy claros y poco variables en el tiempo. Esta metodología me permitirá además volver a fases anteriores para añadir nuevas funcionalidades en caso de que sea necesario o rectificar algunas cuestiones en caso de error.

Sin embargo, durante el desarrollo tendré que tener en cuenta las siguientes desventajas de esta metodología:

- Si se comenten errores y no se detectan en la etapa inmediatamente siguiente es costoso y difícil volver atrás para realizar la corrección.
- El producto no se puede probar hasta el final del desarrollo.

8. EDT:

En el siguiente diagrama se puede ver la descomposición de tareas previstas durante el desarrollo de la aplicación. Como se puede apreciar, las fases principales por las que pasará el proyecto son: DOP (Documento de Objetivos del proyecto), Análisis, Diseño, Implementación, Pruebas, Defensa y Control y seguimiento

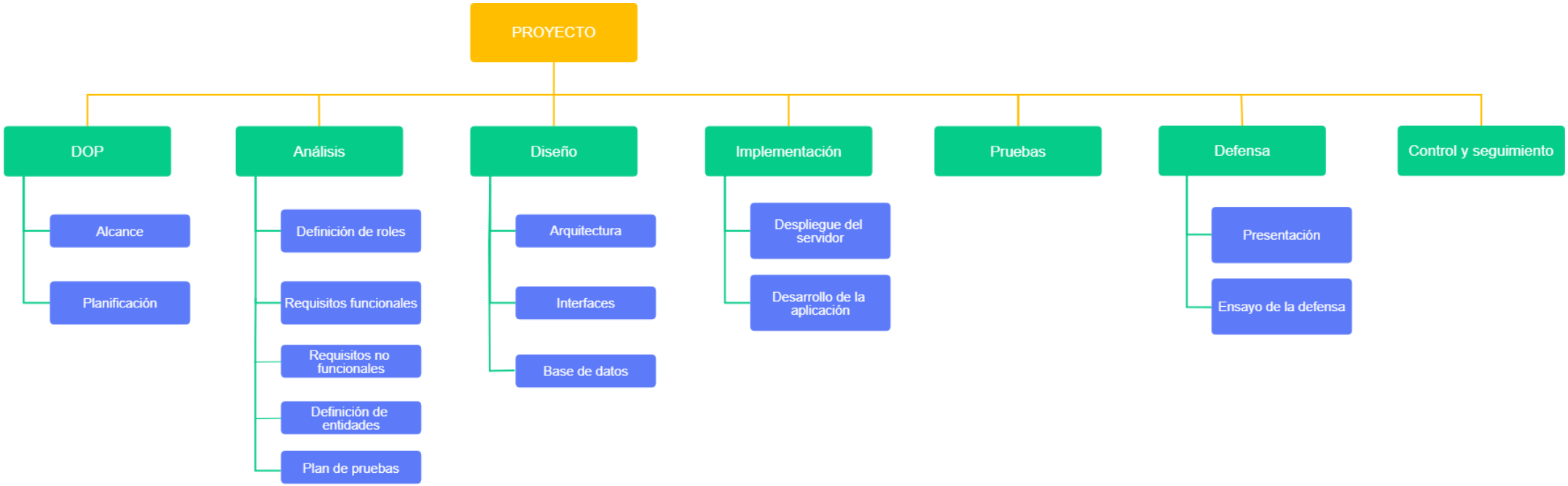


Imagen 1. EDT del proyecto.

9. PLANIFICACIÓN:

Las siguientes tablas (2-3-4) muestran el cronograma del proyecto en el que se definen las duraciones previstas para cada parte del proyecto. Como podemos observar, la planificación parte desde el día 1 de febrero hasta el día 24 de junio.

	Febrero 2019				Marzo			
	4	11	18	25	4	11	18	25
DOP								
Análisis								
Diseño								
Implementación								
Pruebas								
Defensa								
Control y seguimiento								

Tabla 2. Planificación febrero y marzo.

	Abril 2019					Mayo 2019			
	1	8	15	22	29	6	13	20	27
DOP									
Análisis									
Diseño									
Implementación									
Pruebas									
Defensa									
Control y seguimiento									

Tabla 3. Planificación abril y mayo.

	Junio 2019			
	3	10	17	24
DOP				
Análisis				
Diseño				
Implementación				
Pruebas				
Defensa				
Control y seguimiento				

Tabla 4. Planificación junio.

Mi situación durante el desarrollo del proyecto es complicada debido a que estoy trabajando a jornada completa para una empresa en Madrid. Como se puede ver en la tabla 5, el color azul representa las horas dedicadas a la empresa y el color verde representaría el tiempo dedicado al desarrollo del proyecto. No obstante, cabe destacar que es posible que algunos días no se cumplan dichos plazos con lo que tendremos que definir planes de contingencias que veremos más adelante.

	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO	DOMINGO
9:00							
10:00							
11:00							
12:00							
13:00							
14:00							
15:00							
16:00							
17:00							
18:00							
19:00							
20:00							
21:00							
22:00							
23:00							

Tabla 5. Horario semanal.

10. PLAN DE DEDICACIONES:

En este apartado, se tratará la duración estimada para cada una de las tareas que se han descompuesto previamente. Como se puede apreciar, la mayor parte de la carga de horas se encuentra en el desarrollo de la aplicación debido a que es la parte que más dedicación requiere. Hay que programar todas las funcionalidades de la aplicación y realizar pequeñas investigaciones para encontrar librerías adecuadas para determinadas tareas.

	Plan de dedicaciones		
Paquetes de trabajo		Hitos	Horas
E10-1	Alcance del proyecto	8-feb	2
E10-2	Planificación del proyecto	11-feb	10
E20-1	Definición de roles	12-feb	1
E20-2	Requisitos funcionales	15-feb	1
E20-3	Requisitos no funcionales	15-feb	1
E20-4	Definición de entidades	20-feb	1
E20-5	Plan de pruebas	25-feb	1
E30-1	Diseño de la arquitectura	27-feb	1
E30-2	Diseño de interfaces	7-mar	6
E30-3	Diseño de la base de datos	11-mar	3
E40-1	Despliegue del servidor	18-mar	15
E40-2	Aplicación web	3-jun	230
E50-1	Pruebas	17-jun	3
E60-1	Presentación	20-jun	2
E70-1	Documento de control y seguimiento	20-jun	8
	Total de horas:		285

Tabla 6. Horas.

11. PLAN DE CONTINGENCIAS:

En el caso de que los horarios planificados no se puedan cumplir se tratará de redistribuir la carga del día perdido entre el resto de los días de la semana. No obstante, si la desviación producida ha sido considerable, se utilizarán los días festivos como semana Santa, el día del trabajador u otros puentes disponibles para volver a los plazos establecidos.

Además, las tardes de los viernes (que están libres en la anterior planificación), podrán ser usadas para recuperar las horas que se hayan perdido durante la semana o incluso para hacer un esfuerzo extra si fuese necesario.

En el caso de que fueran necesarias más horas debido a algún retraso considerable, se pospondrán las horas destinadas para la preparación de la defensa a partir del día 24 de junio hasta la defensa del proyecto.

CAPÍTULO 2. ANÁLISIS

Experimenta-Unirioja es una actividad organizada por el Decanato de la Facultad de Ciencia y Tecnología de la Universidad de la Rioja. Dicha actividad está formada por un conjunto de talleres que son organizados por coordinadores de cada una de las titulaciones de la Facultad que se involucran en el evento.

Inicialmente, se realizaba una difusión mediante correo electrónico y se anotaban las reservas a mano. Poco a poco, el número de solicitudes fue creciendo hasta que llegó un punto en el que se tuvo que implantar un formulario web para realizar las reservas. Sin embargo, el sistema actual sigue siendo insuficiente ya que los coordinadores no pueden configurarlo cada año, necesitan ayuda de un perfil informático.

Los talleres suelen durar alrededor de unas dos horas y, dependiendo de cada titulación, suele haber entre uno y cuatro talleres por semana. Estos se suelen realizar durante los meses de marzo y abril, aunque el sistema que se creará permitirá poder extender los cursillos durante el tiempo que se desee.

Las reservas las realizan los profesores de los centros educativos, es decir, es un profesor el que realiza la reserva para todos sus alumnos. Durante los años anteriores, Experimenta-Unirioja ha contado con cuatro tipos de talleres:

- Químico por un día (Grado en Química).
- Matemática creativa (Grado en Matemáticas).
- Surfeando en la ingeniería informática (Grado en Ingeniería Informática).
- Descubriendo las ciencias agroalimentarias (Grado en Ingeniería Agrícola).

Para la realización de la recogida de requisitos y para todo el proceso de análisis y diseño seguiré UML[1]. Utilizaré diagramas de casos de uso y diagramas de clases para explicar las principales funcionalidades del sistema.

1. DEFINICIÓN DE ROLES.

En esta aplicación tenemos cuatro tipos de roles diferenciados principalmente en dos tipos:

- **Usuario registrado**. Diferenciamos tres tipos de usuarios registrados:
    - El **administrador**, es el encargado de gestionar los permisos de la aplicación y de la creación de los usuarios encargados de gestionar los cursillos (organizadores). Además, podrá realizar todas las funcionalidades de los otros usuarios de la aplicación.
    - El **organizador**, es el usuario que podrá realizar la gestión de los talleres, así como ver estadísticas de los talleres creados y la gestión de las reservas.
    - El **solicitante**, es el usuario que podrá reservar plazas en un taller, pero al haberse registrado previamente, le será más fácil y rápido hacerlo. Además, podrá tener la posibilidad de tener una visión general de las reservas realizadas.
- **Usuario no registrado**. Es una persona que entra en la aplicación web para reservar plazas en un taller. Este no se registra y la aplicación no guardará información sobre él.

[1] Unified Modeling Language, El lenguaje unificado de modelado es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

2. DEFINICIÓN DE REQUISITOS FUNCIONALES.

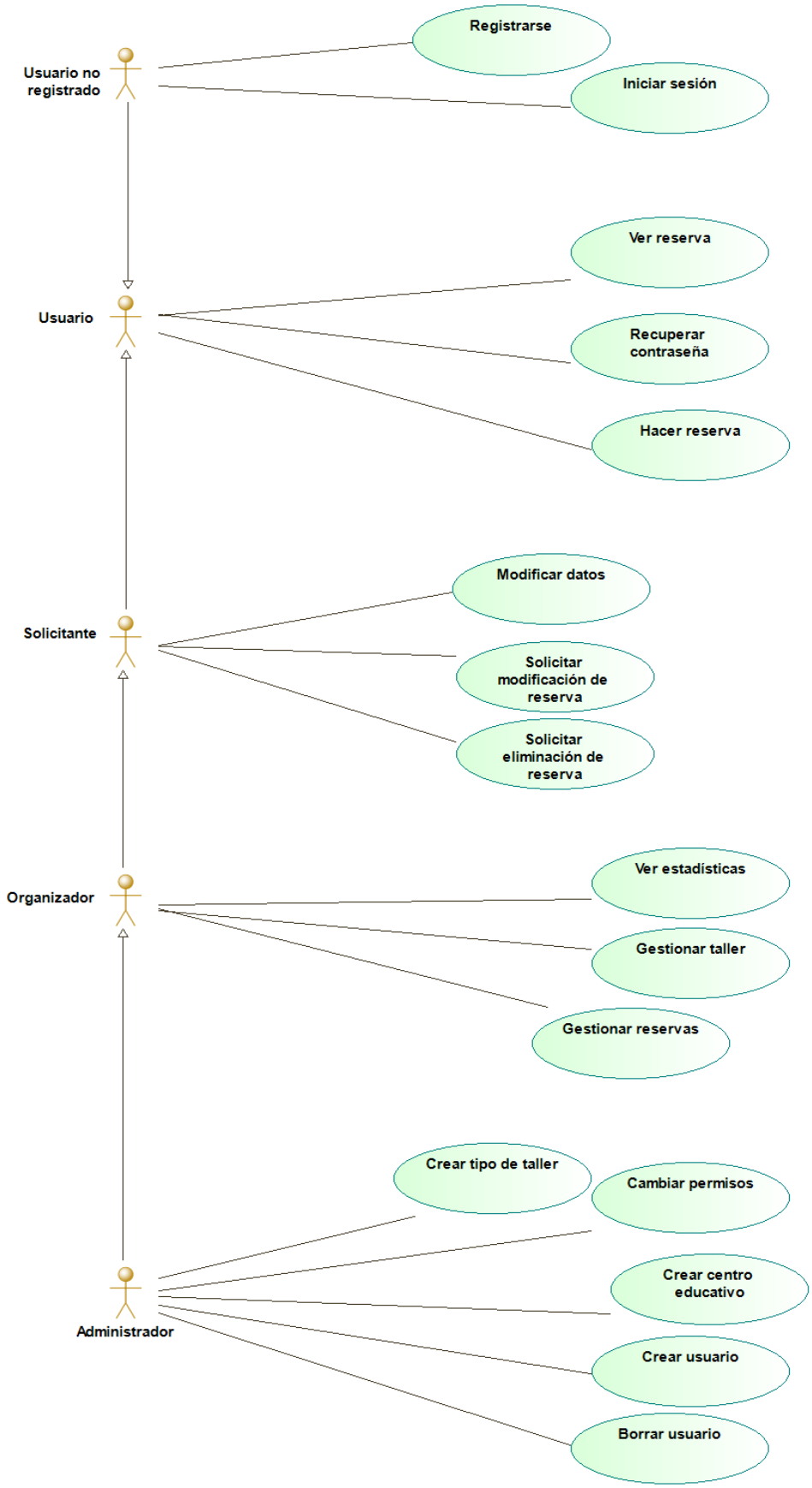


Imagen 2. Diagrama de casos de uso

REQUISITOS FUNCIONALES	
Nombre: Crear usuario	Actor: Administrador
Precondición: estar autenticado como administrador.	
Postcondición: Tras el proceso se debe crear un usuario nuevo.	
Descripción: el administrador rellenará un formulario con los datos necesarios para crear un usuario y le asignará el tipo de permisos que crea conveniente.	
Nombre: Borrar usuario	Actor: Administrador
Precondición: estar autenticado como administrador.	
Postcondición: Tras el proceso se debe borrar un usuario de la BD.	
Descripción: aparecerá un listado de todos los usuarios del sistema y el administrador tendrá la posibilidad de eliminar cualquier usuario.	
Nombre: Cambiar permisos	Actor: Administrador
Precondición: estar autenticado como administrador.	
Postcondición: Tras el proceso un usuario tendrá un nuevo permiso.	
Descripción: aparecerá un listado de todos los usuarios del sistema y el administrador tendrá la posibilidad de editar sus datos personales y modificar sus permisos.	
Nombre: Crear tipo de taller (actividad)	Actor: Administrador
Precondición: estar autenticado como administrador.	
Postcondición: Tras el proceso se habrá añadido una nueva actividad al sistema.	
Descripción: se abre un formulario para introducir el nombre de la actividad y tras aceptar se crea en la BD.	
Nombre: Crear centro educativo	Actor: Administrador
Precondición: estar autenticado como administrador.	
Postcondición: Tras el proceso se habrá añadido un nuevo centro al sistema.	
Descripción: se abre un formulario para introducir el nombre del centro y sus datos y tras aceptar se crea en la BD.	
Nombre: Gestionar taller	Actor: Organizador / Administrador
Precondición: estar autenticado como organizador o como administrador	
Postcondición: al final se ha debido realizar una modificación, una eliminación o una creación de un taller.	
Descripción: se elegirá una de las opciones de gestión de talleres (creación, edición o eliminación) del menú de navegación y se procederá a su ejecución.	
Nombre: Gestionar reservas	Actor: Organizador / Administrador
Precondición: estar autenticado como organizador o administrador.	
Postcondición: se habrá hecho una modificación, eliminación o creación de una reserva.	
Descripción: se elegirá una de las opciones de gestión de reservas del menú de navegación y se procederá a su realización.	
Nombre: Ver estadísticas	Actor: Organizador / Administrador
Precondición: estar autenticado como organizador o administrador.	
Postcondición: se abrirá una página que mostrará las estadísticas disponibles.	
Descripción: el usuario seleccionará la opción de estadísticas en el menú de navegación y se abrirá una página con las estadísticas disponibles.	
Nombre: Modificar datos	Actor: Organizador / Administrador / Solicitante
Precondición: estar autenticado como organizador, administrador o solicitante.	
Postcondición: se modificarán los datos almacenamos del usuario.	
Descripción: el usuario selecciona la opción de modificar sus datos personales en el icono de la cabecera y se abre un formulario para modificar sus datos.	

Nombre: Solicitar modificación de reserva	Actor: Organizador / Administrador / Solicitante
Precondición: estar autenticado como organizador, administrador o solicitante.	
Postcondición: se enviará una notificación al organizador del taller correspondiente avisando de la petición.	
Descripción: el usuario seleccionará la opción en el menú de navegación, seleccionará la reserva que quiere modificar e indicará los cambios que desea.	
Nombre: Solicitar eliminación de reserva	Actor: Organizador / Administrador / Solicitante
Precondición: estar autenticado como organizador, administrador o solicitante.	
Postcondición: se enviará una notificación al organizador del taller correspondiente indicando que se desea eliminar la reserva.	
Descripción: el usuario seleccionará la opción en el menú de navegación, seleccionará la reserva que quiere eliminar y confirmará la petición.	
Nombre: Ver reserva	Actor: Organizador / Administrador / Solicitante / Usuario no registrado
Precondición:	
Postcondición: se mostrarán los datos de la reserva solicitada.	
Descripción: se indicará el identificador de la reserva mediante un formulario y se mostrarán los datos correspondientes a ese identificador.	
Nombre: Recuperar contraseña	Actor: Organizador / Administrador / Solicitante
Precondición: estar autenticado como organizador, administrador o solicitante.	
Postcondición: el usuario podrá restaurar su contraseña.	
Descripción: el usuario solicitará la recuperación de la contraseña a través del login. Recibirá en su correo un enlace desde el que podrá crear una nueva contraseña.	
Nombre: Hacer reserva	Actor: Organizador / Administrador / Solicitante / Usuario no registrado
Precondición: estar autenticado como organizador, administrador, solicitante o usuario no registrado.	
Postcondición: el usuario habrá realizado una reserva para un taller del calendario.	
Descripción: el usuario hará clic sobre uno de los talleres mostrados en el calendario. Se abrirá un formulario para hacer la reserva.	
Nombre: Registrarse	Actor: Usuario no registrado
Precondición:	
Postcondición: se creará un usuario en la aplicación.	
Descripción: desde la página de registro se rellenará un formulario con los principales datos del usuario y tras hacer clic en registrarse se creará el usuario.	
Nombre: Iniciar sesión	Actor: Usuario no registrado
Precondición:	
Postcondición: se redireccionará a la página principal interna de la aplicación.	
Descripción: el usuario introducirá sus credenciales y en caso afirmativo entrará en la parte privada de la aplicación.	

Tabla 7. Casos de uso

3. DEFINICIÓN DE REQUISITOS NO FUNCIONALES.

Los requisitos no funcionales pensados para este proyecto son:

- Desarrollo en el entorno Visual Studio Code
- Utilización de la tecnología Angular 6 para el desarrollo del Front de la aplicación.
- Utilización de NodeJS para la parte Back de la aplicación.
- Creación y gestión de las bases de datos mediante la herramienta MongoDB Compass.
- Se utilizará un servidor con sistema operativo CentOS Linux 7 para el despliegue de la aplicación.
- Las vistas de la aplicación deberán verse correctamente en los navegadores más importantes (Google Chrome, Mozilla Firefox, Microsoft Edge y Safari).

4. PLAN DE PRUEBAS.

Una vez implementadas las funcionalidades del proyecto, se deberá hacer una comprobación sobre los siguientes procesos:

- **Login de usuario.** Se deberá comprobar que un usuario registrado puede acceder correctamente a la aplicación mediante su usuario y su contraseña. De la misma forma, también tendrá que comprobarse que un usuario con credenciales incorrectas no pueda entrar de ninguna forma al interior de la aplicación.
- **Alta de taller.** Los organizadores podrán crear talleres con los datos que ellos quieran introducir. Se deberá comprobar que la creación se visualiza posteriormente y que todo ha quedado reflejado en la base de datos. Se debe hacer la misma comprobación con el rol de administrador ya que también debe tener la misma funcionalidad.
- **Baja de taller.** En cualquier momento los organizadores podrán cancelar uno de los talleres creados. Se deberá comprobar que la eliminación se ha hecho con éxito y que se ha borrado la información de la base de datos. Misma comprobación en el caso del administrador.
- **Modificar taller.** Los organizadores podrán hacer modificaciones sobre los talleres creados. Habrá que crear un taller tanto como organizador como administrador y comprobar que todos los cambios quedan reflejados en la base de datos.
- **Creación de reserva.** Se hará una reserva para comprobar que todos los datos introducidos por el usuario se guardan correctamente, así como que el número de plazas disponibles para el taller se vean reducidas. También habrá que comprobar que el usuario que realiza la reserva recibe una confirmación a través de su correo electrónico y que el organizador del taller también recibe una notificación.
- **Solicitud de eliminación de reserva.** Un usuario puede solicitar la eliminación de una de sus reservas. Se debe comprobar que la solitud queda registrada perfectamente en la base de datos y que el organizador recibe un correo electrónico de la solicitud.
- **Solicitud de modificación de reserva.** Un usuario puede solicitar la modificación de una de sus reservas. Se debe comprobar que la solicitud queda registrada perfectamente en la base de datos y que el organizador recibe un correo electrónico de la solicitud.
- **Gestión de centros.** El administrador puede añadir, editar o eliminar centros educativos al sistema. Se deberá crear un centro, editarlo y eliminarlo, viendo cómo se van aplicando todos los cambios en la base de datos.

- **Cierre de sesión.** Se comprobará que el usuario puede hacer el log-out y que la sesión se cierra correctamente. La aplicación no deberá abrirse si hacemos clic en el botón de “ir atrás” disponible en el navegador, ni acceder a través de cualquier ruta.
- **Creación de usuario.** Se creará un usuario y se comprobará que los datos se han guardado correctamente, así como que la funcionalidad de este es la adecuada y funciona correctamente.
- **Eliminación de una reserva.** Se eliminará una reserva, comprobando que desaparece la misma de la BD y que se restablecen las plazas reservadas en el taller correspondiente.
- **Inserción de logos.** Desde el panel del administrador es posible configurar los logos de la página de inicio de la aplicación. Cuando se inserta un logotipo es necesario comprobar que este se almacena correctamente en el servidor y que se muestra en la página principal.
- **Asignación de colores.** En el momento de creación de una actividad es posible asignar un color a esa actividad. Hay que comprobar que el color se guarda en el formato correcto y que después los talleres correspondientes a esa actividad se pintan con el color en el calendario.
- **Gestión de actividades.** Un administrador puede crear distintas actividades. Estas actividades se tienen que almacenar correctamente en la BD. Además, deberán aparecer disponibles al crear un usuario para ser asignadas y en la creación de un taller.
- **Cambiar contraseña**. Desde el perfil del usuario es posible cambiar la contraseña. El sistema deberá comprobar si las dos contraseñas nuevas introducidas coinciden. Si lo hacen se verificará que la contraseña antigua es correcta y caso afirmativo se actualizará la contraseña en la BD. Será necesario salir de la aplicación e intentar entrar de nuevo para comprobar que el cambio es efectivo.

CAPÍTULO 3. DISEÑO:

1. ARQUITECTURA

Este proyecto tiene una arquitectura diferenciada principalmente en dos partes. Una parte es el frontal con el que interactúa el usuario, desde donde se realizan las inscripciones a los talleres y el resto de las funcionalidades. Esta parte está programada con el lenguaje TypeScript (Angular 6).

La otra parte es el back de la aplicación. Este está formado por una API REST que recoge todas las peticiones que se realicen desde el lado del cliente y una base de datos que almacena toda la información necesaria para el funcionamiento del sistema. La API está construida con el lenguaje JavaScript (NodeJS) y la base de datos es una base de datos NoSQL, concretamente MongoDB.

Los dispositivos harán peticiones al servidor que se encargará de que lleguen al servidor de aplicaciones (apache). En función de la petición, la aplicación devolverá directamente una vista o recurrirá a la base de datos para obtener o guardar algún dato antes de proceder a la respuesta. Angular sigue el MVC [2]donde la vista es el conjunto de archivos HTML y CSS de un componente y el controlador es el archivo typescript que hace de intermediario entre el servidor y el frontal.

2. DIAGRAMA DE CLASES.

En la siguiente figura (imagen 3) podemos ver el diagrama de clases que representa nuestro sistema. Está formado por doce clases.

Seis de ellas corresponden a la gestión de los usuarios. La clase *Usuario* representa a todos los usuarios de la aplicación y contiene los campos comunes entre ellos (email, id, teléfono y centro). Un usuario puede ser de dos tipos, un *Usuario Registrado* o un *Usuario No Registrado*. La diferencia la encontramos en la cantidad de atributos del objeto usuario. El *Usuario Registrado* tendrá tres campos más (*password*, nombre y rol). A su vez, un *Usuario Registrado* puede ser de tres tipos dependiendo del valor de su atributo Role; *Organizador*, *Administrador* o *Solicitante*. El organizador tendrá un campo más llamado Tipo que representará el tipo de talleres con los que puede interactuar en la aplicación.

En caso de que alguno de los usuarios olvide su contraseña será posible recuperarla. Es por ello por lo que hay una clase llamada *Recuperación Contraseña*. Esta clase representa una solicitud de restauración de contraseña que vincula el email del usuario con un UUID [3] y con una fecha de validez. Esto permitirá mediante un enlace recibido en el correo electrónico cambiar la contraseña.

La clase Taller permitirá gestionar los talleres de la aplicación, contendrán la información básica que los caracteriza (nombre, tipo, fecha…) así como el usuario que los organiza. La clase *Reserva* recoge todas las inscripciones en los talleres Experimenta. Por ello, tiene una referencia al taller, otra al usuario que realiza la reserva y el número de plazas que se bloquean.

[2] MVC: Modelo Vista Controlador.

[3] UUID: un identificador único universal. Es un número de 16 bytes expresado mediante 32 dígitos hexadecimales.

En alguna ocasión, será posible que un usuario registrado pueda solicitar una modificación o eliminación de una reserva. De esta necesidad surge la clase *Solicitud*, que permitirá controlar estas situaciones y almacenar el tipo de solicitud, la fecha, la reserva, el estado e información de esta.

La clase Tipo representa las distintas actividades que forman los talleres Experimenta. Un taller pertenece a una actividad (tiene un tipo). Y de la misma forma, un organizador tiene una actividad asignada sobre la que tiene control.

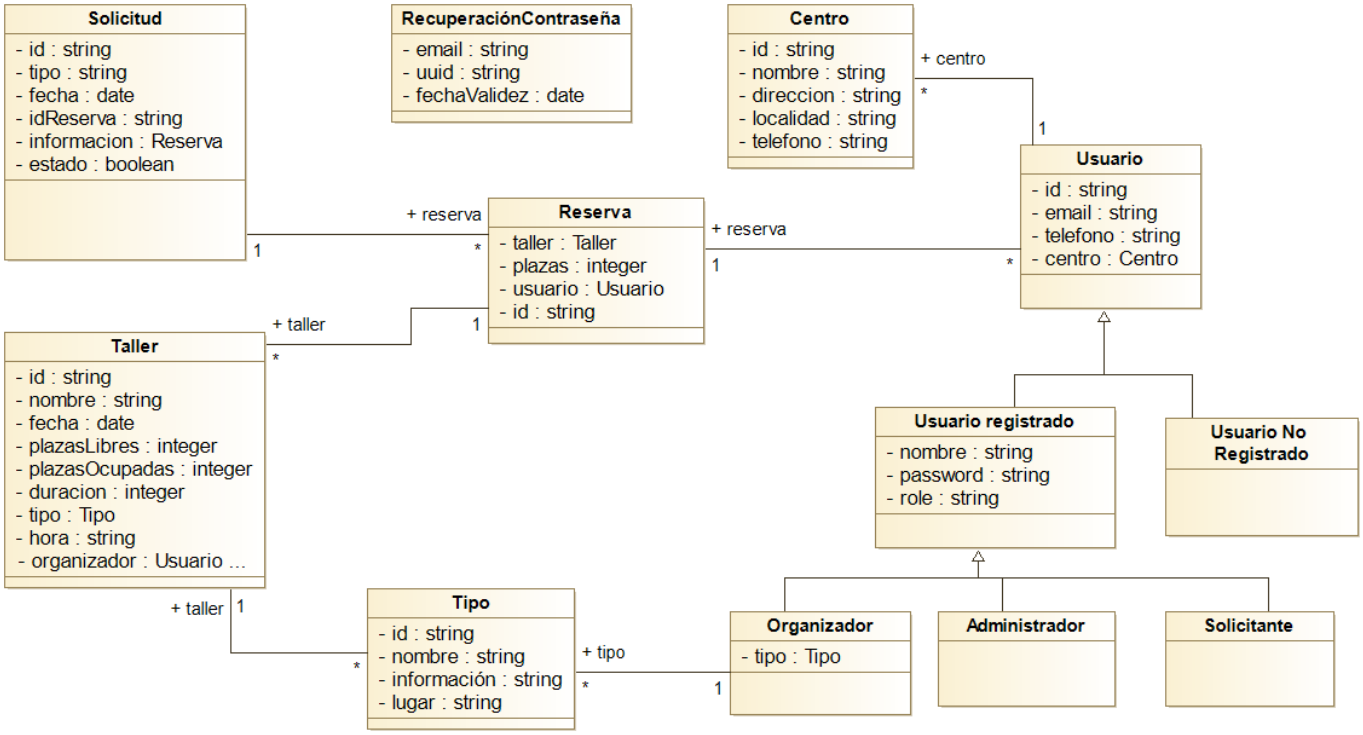


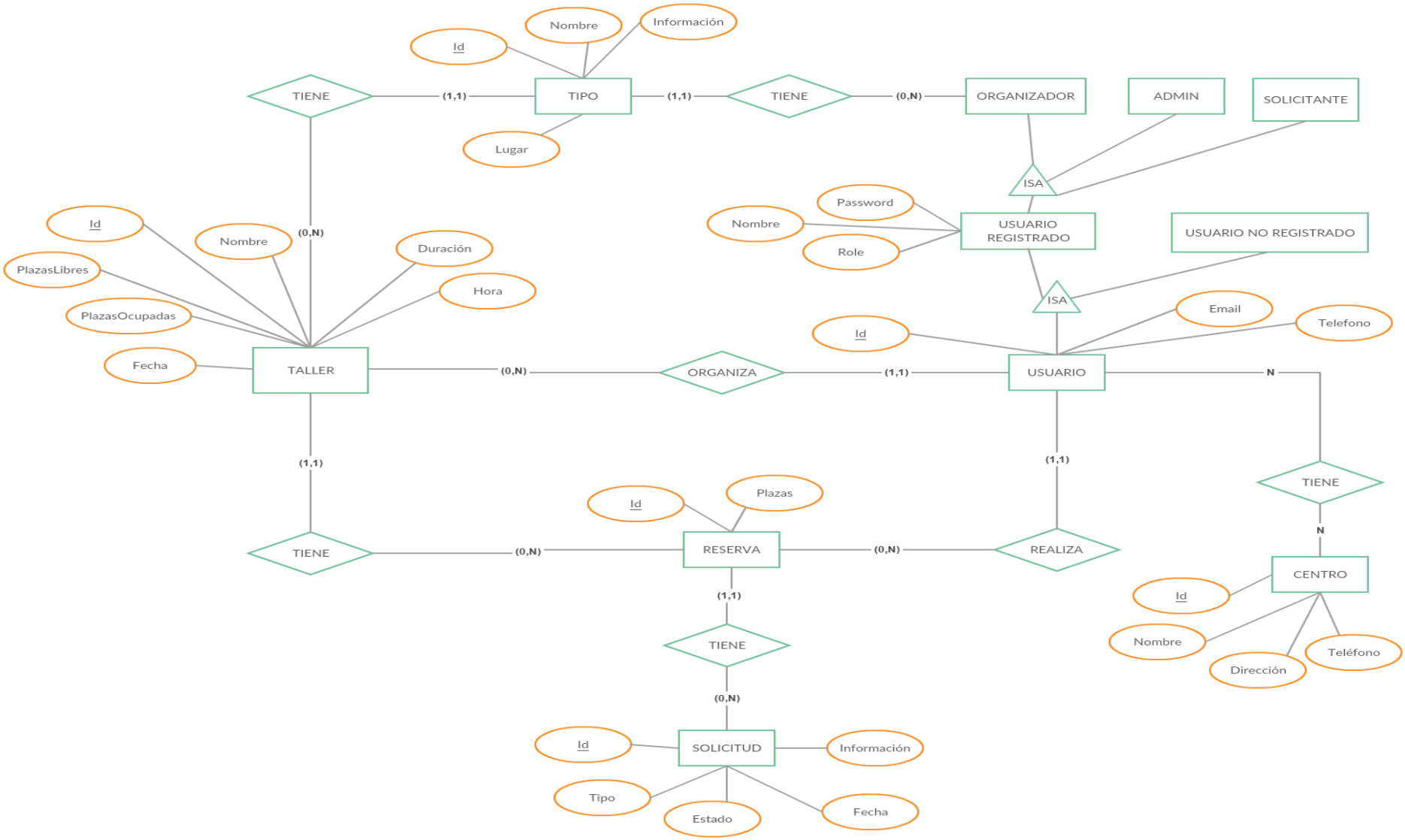
Imagen 3. Diagrama de clases

3. DISEÑO DE LA BASE DE DATOS:

El diagrama que veremos a continuación es el diagrama E/R (Entidad / Relación) que representa como las distintas entidades de nuestra aplicación se relacionan entre sí. Este diagrama nos servirá para diseñar posteriormente la base de datos.

No obstante, es posible que en el posterior diseño aparezcan nuevas entidades que no están contempladas en el siguiente diagrama.

Imagen 4. Diagrama E/R



A continuación, se muestra la transformación realizada del diagrama E/R a las tablas que representan la base de datos. La transformación, no se ha realizado mediante el proceso de normalización del diseño de las bases de datos relacionales, debido a que para este proyecto se va a utilizar una base de datos NoSQL, orientada a documentos (MongoDB).

TALLER:

Id	Nombre	Actividad	Fecha	Hora	Duración	PlazasLibres	PlazasOcupadas

Tabla que representa un taller de una actividad. Los campos *id*, *nombre*, *actividad* y *hora* son de tipo *String*. La *fecha* será de tipo *Date* y, la *duración*, las *plazasLibres* y las *plazasOcupadas* serán de tipo *Number*.

ACTIVIDAD:

Id	Nombre	Información	Lugar

Una actividad es un tipo de talleres, por ejemplo "Surfeando en la Ingeniería Informática". Está formado por un *id*, un *nombre*, una breve *descripción* y un *lugar*. Todos los campos serán de tipo *string*.

USUARIO:

Id	Email	Telefono	Centro	Role	Contraseña	Nombre

La tabla Usuario representará a los usuarios de la aplicación. Estos podrán tener distintos roles que estarán definidos mediante el campo *Role* (*String*). Todos los usuarios tendrán que tener de forma obligatoria un id, un email, un teléfono y un centro, todos de tipo String. En el caso de que no tengan el resto de los campos (*contraseña*, *nombre o rol*) significará que son usuarios no registrados y solo podrán hacer reservas mediante el calendario externo. En el otro caso si podrán entrar a la aplicación con el rol que tengan. La fecha de nacimiento será de tipo *Date*. La *contraseña* y el *nombre* serán de tipo *String*.

RESERVA:

Id	IdTaller	IdUsuario	Plazas

El esquema reserva sirve para representar las reservas realizadas para un usuario en un determinado taller. Tendrá un id propio, el id del taller que se reserva y el id del usuario para el que se reserva. Otro campo de tipo *number* indicará el número de plazas reservadas.

SOLICITUD:

Id	IdReserva	Información	Fecha	Estado	Tipo

El esquema solicitud representa una petición del usuario para eliminar o editar una reserva. En concreto, el campo tipo indica si es una solicitud de modificación o cancelación (*string*). El campo estado (*boolean*) indica si se ha resuelto o no la petición. La *fecha* indica el momento en el que se ha realizado la petición y la información es una breve descripción (*string*) que el usuario ha escrito para realizar la solicitud.

RECUPERACIÓN_CONTRASEÑA:

id	Email	UIID	FechaValidez

Este es el esquema que se utiliza cuando se hace una solicitud para restaurar la contraseña. El email (*string*) representa al usuario. El UIID es un identificador único que permite verificar la seguridad del cambio de contraseña. Solo el usuario recibirá un email con un enlace que contiene ese código. La fecha de validez indica el periodo durante el cual el UUID es válido.

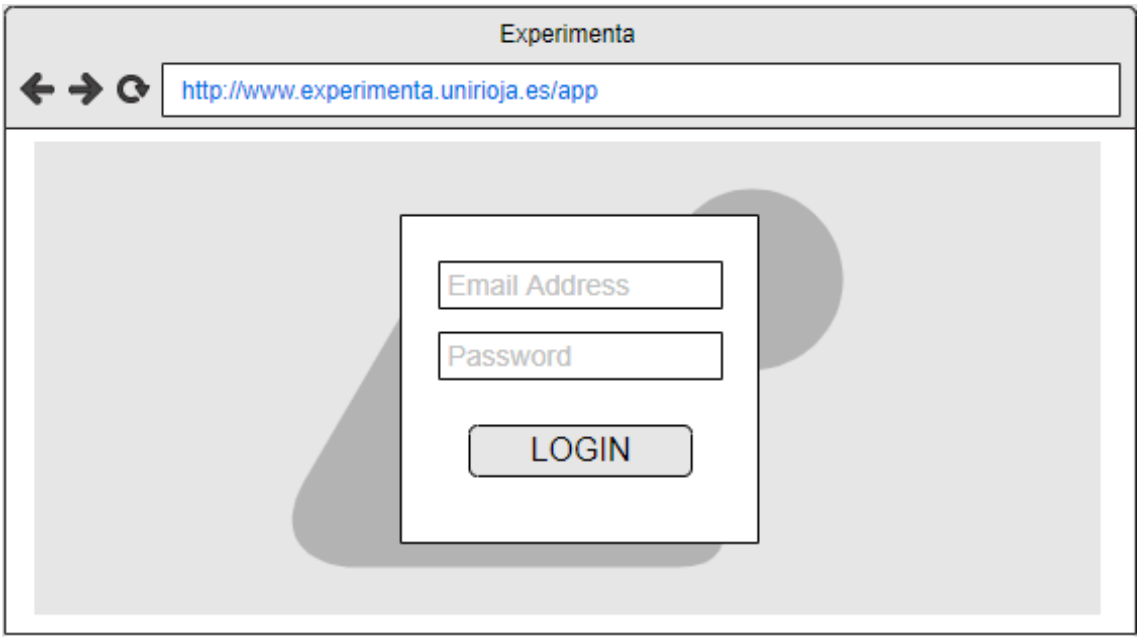
CENTRO:

id	Nombre	Dirección	Teléfono	Localidad

El esquema Centro representa los centros escolares que están disponibles en la aplicación. Se componen de un id, un nombre, una dirección, un teléfono y una localidad (todos de tipo *string*).

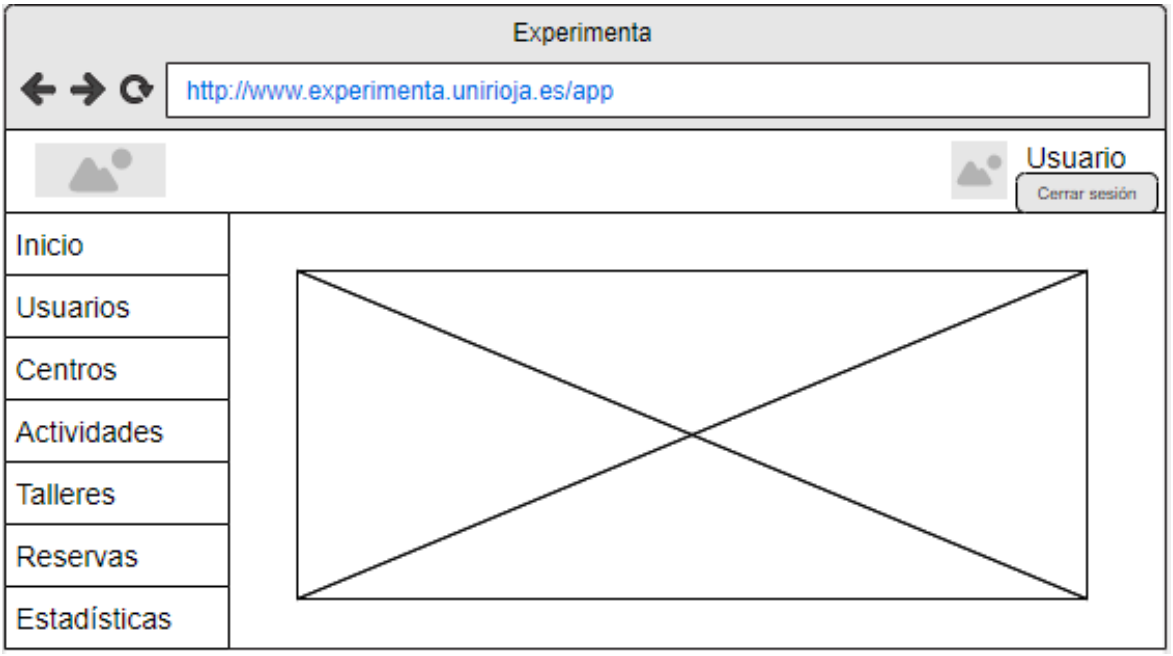
4. INTERFACES.

Para comenzar, será necesario una interfaz para acceder a la aplicación. En ella se realizará la autenticación del usuario, la creación de una cuenta o la recuperación de contraseña.

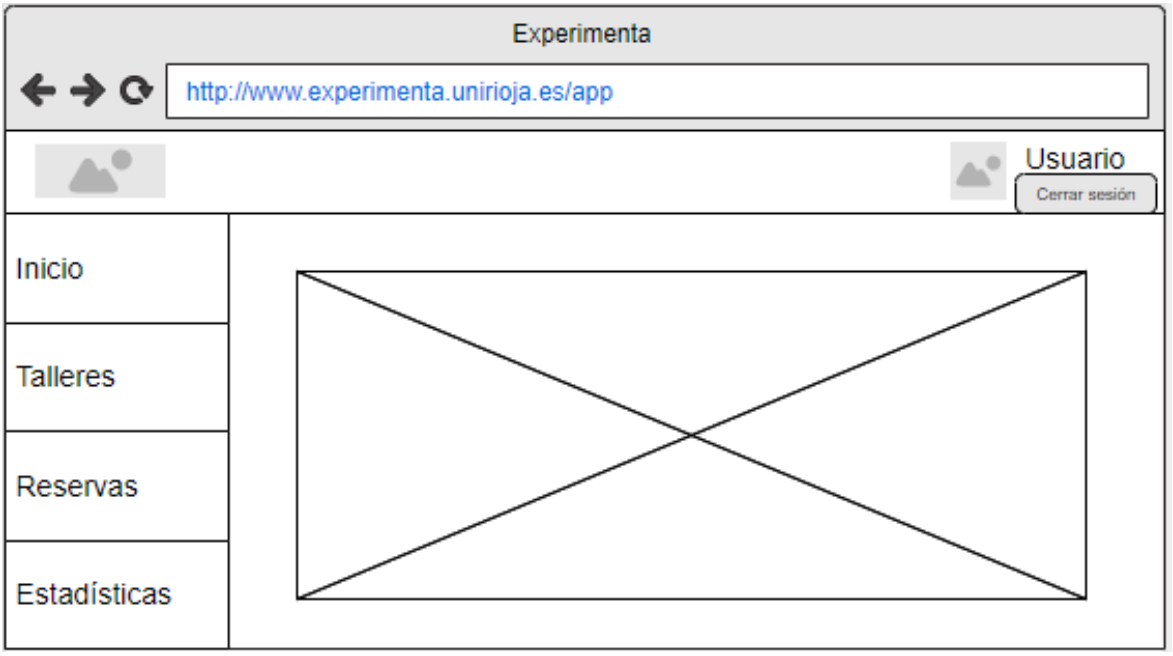


El interior de la aplicación tendrá siempre un mismo estilo. Una cabecera con el logo de la universidad, el apartado para el cierre de sesión y el perfil de usuario, y una barra de navegación lateral desde donde estarán accesibles todas las funcionalidades de la aplicación.

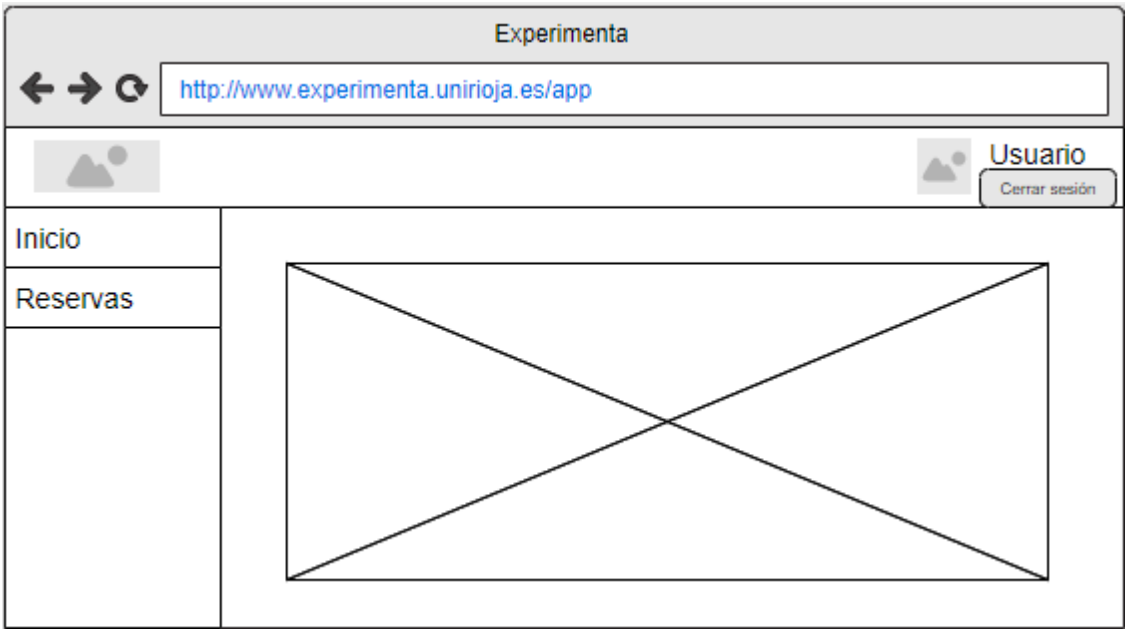
Interfaz de administrador:



Interfaz de organizador:



Interfaz del Solicitante:



Como se puede ver en los bocetos anteriores, la estructura de la aplicación es siempre la misma. Dependiendo de cada usuario hay más o menos opciones disponibles en la barra de navegación lateral y todas las funcionalidades se despliegan sobre el hueco central disponible.

5. Diseño de la API

Para el funcionamiento de la aplicación se creará una API REST en un servidor que se encargará de enviar y recibir datos desde el frontal de la aplicación hasta la base de datos. Para su funcionamiento se definirán un conjunto de rutas con las distintas necesidades de la BD. Todas las rutas vendrán definidas por "/api/":

- /api/reservas: para gestionar las reservas.
- /api/tipos: para la gestión de las actividades.
- /api/centros: creación, eliminación y modificación de centros escolares.
- /api/solicitudes: peticiones de edición y eliminación de reservas.
- /api/restaurarPassword: para restaurar la contraseña.
- /api/usuarios: para la creación de usuarios y la autenticación.
- /api/forgotPassword: para solicitar el envío de correo de recuperación.

CAPÍTULO 4. IMPLEMENTACIÓN

En este capítulo se abordará la construcción del sistema definido durante todo el proyecto. Como se ha visto anteriormente, se diferencian dos partes principales para el desarrollo; el frontal de la aplicación que interactúa con el usuario y la API REST que gestiona todas las peticiones y conexiones con la base de datos.

1. CONSTRUCCIÓN DEL FRONT-END.

Para la construcción del frontal se ha utilizado Angular 6. Es una plataforma para el desarrollo de aplicaciones web para móviles y escritorio. Se caracteriza principalmente por su rapidez ya que alcanza la mayor velocidad posible en las plataformas web actuales. Además, Angular permite construir de forma rápida y sencilla determinados componentes disponibles en librerías. Es un framework SPA[4], una aplicación web de una sola página, es decir, permite no tener que recargar toda la página con cada interacción del usuario si hay partes que no se ven alteradas.

Su principal objetivo es potenciar las aplicaciones basadas en el Modelo Vista Controlador (MVC) para permitir que el desarrollo y las pruebas sean mucho más fáciles de realizar. Otro de los propósitos que tiene angular es separar totalmente la parte de *front-end* y *back-end*. Solo se encargará de la parte del cliente a diferencia de otras aplicaciones monolíticas en las que ambas partes estaban en la misma aplicación (aunque estuvieran separadas en capas). Esto permite que los proyectos sean mucho más compresibles además de proporcionar una mayor velocidad en el rendimiento de las aplicaciones.

Otra de las grandes ventajas de este framework es que ha sido desarrollado por Google, lo que significa que tiene una gran comunidad de desarrolladores que lo mantienen y lo mejoran cada día. Esto es muy importante a la hora de desarrollar, ya que permite encontrar la solución a muchos de los problemas con los que te enfrentas.

1.1.1 Módulos:

Angular usa TypeScript, HTML y CSS para crear las aplicaciones. Los bloques básicos de construcción de una aplicación Angular son los ***ngModules*** (módulos), que proporcionan un contexto para los **componentes**. Todas las aplicaciones tienen que tener como mínimo un módulo (el root-module). Para este proyecto solo se ha utilizado un par de módulos. Veamos un ejemplo, el AppModule:

```
@NgModule({
  declarations: [
    AppComponent,CabeceraComponent,NavBarComponent,CalendarioComponent,
      HomeComponent,CrearTallerComponent,ModificarTallerComponent,ConstruirFe
chasPipe,DialogEditComponent,EliminarTallerComponent,DialogEliminarTallerComp
onent,VerReservasComponent,DialogReservasComponent,ReservaRegistradoComponent
…
],

imports: [
   BrowserModule,BrowserAnimationsModule,AppRoutingModule,MatSidenavModule,
MatExpansionModule,MatCheckboxModule,FullCalendarModule,MatDatepickerModule,M
atFormFieldModule,MatNativeDateModule,MatInputModule…
  ],
```

[4] SPA: Single Page Aplication

```
providers: [MatDatepicker, DatePipe, {provide: HTTP_INTERCEPTORS, useClass:
AuthInterceptor, multi:true}],
  bootstrap: [AppComponent],
  entryComponents: [ DialogEditComponent, DialogEliminarTallerComponent,
DialogReservasComponent, DialogEliminarReservaComponent,
DialogEliminarUsuarioComponent,DialogSolicitarEdicionComponent,
DialogEditarUsuarioComponent…
  ]
```

Imagen 5. App Module

1.1.2 Componentes:

Los componentes son los elementos que definen las vistas, es decir, son un conjunto de pantallas que Angular intercambia para modificar el programa con lógica y datos. Un componente está formado principalmente por tres tipos de archivos:

- Un archivo HTML para definir la parte visual de la vista

```
<app-cabecera></app-cabecera>
<app-nav-bar></app-nav-bar>
<div class="contenido">
  <div class="fondo">
    <img src="assets/images/fondoInicio.jpg" *ngIf="imagenId==undefined">
    <img [src]="'http://localhost:3000/api/logos/inicio/'+imagenId"
*ngIf="imagenId!=undefined">
      <div class="placaFondo">
          <h1>Bienvenido a la Aplicación Experimenta Unirioja</h1>
          <p *ngIf="usuario=='admin'">Gestiona y controla todos los datos
de tus usuarios, centros, talleres y reservas</p>
          <p *ngIf="usuario=='organizador'">Gestiona y controla todos los
datos de tus talleres y reservas</p>
          <p *ngIf="usuario=='visitor'">Gestiona y controla todos los
datos de tus reservas</p>
        </div>
  </div>

</div>
```

Imagen 6. HTML de un componente

- Un archivo CSS para añadir los estilos a la parte visual

```
.fondo img {
    position: absolute;
}

.placaFondo {
  background: #ebebeb;
  position: absolute;
  opacity: 0.8;
  margin-left: 4%;
  padding: 2% 0% 2% 4%;
}
```

Imagen 7. CSS de un componente

- Un archivo de extensión .ts (lenguaje typescript) que contiene la lógica de la vista.

```
import { Component, OnInit } from '@angular/core';
import { PetitionService } from '../servicios/petition.service';

@Component({
  selector: 'app-inicio-app',
  templateUrl: './inicio-app.component.html',
  styleUrls: ['./inicio-app.component.css']
})
export class InicioAppComponent implements OnInit {

  usuario : string;
  imagenId : string;
  constructor(private petitionService: PetitionService) { }

  ngOnInit() {
    this.usuario=localStorage.getItem('role');
    if(this.usuario=='organizador')
    {

this.petitionService.getTipoId(localStorage.getItem('tipo')).then(tipo=> {
        console.log(tipo);
        this.imagenId=tipo['nombreImagen'];
      });
    }
  }

}
```

Imagen 8. Controlador de un componente

Las capturas anteriores muestran el componente que genera la página de inicio de la aplicación que aparece después de iniciar sesión. El resto de los componentes de la aplicación están formados por los mismos archivos y funcionan todos de la misma forma.

Hay un componente para la cabecera y otro componente para el menú de navegación lateral. Estos se utilizan desde el resto de los componentes, lo que nos permite reutilizar el código y ganar velocidad en el proceso de carga. Por lo tanto, en la mayoría de los componentes se cambia únicamente el contenido del espacio restante.

Para que la vista cambie un componente por otro se utiliza un módulo que se llama *app-routing.module.* En su interior, se define una ruta para cada página de la aplicación y se indica el componente que se generará al introducir la dirección lo que permitirá realizar los cambios en la aplicación para transformar las vistas.

```
import { CrearTipoTallerComponent } from './admin/crear-tipo-taller/crear-tipo-
taller.component';
import { PerfilUsuarioComponent } from './perfil-usuario/perfil-usuario.component';

const routes: Routes = [
  { path: '', component: HomeComponent  },
  { path: 'login', component: LoginComponent  },
  { path: 'olvidarPassword', component: OlvidarPasswordComponent  },
  { path: 'recuperarPassword/:uuid', component: RecuperarPasswordComponent  },  {
path: 'verReserva/:id', component: VerReservaComponent  },
  { path: 'home', component: InicioAppComponent  },
  { path: 'calendario', component: CalendarioHomeComponent  },
  { path: 'calendarioApp', component: CalendarioComponent  },
  { path: 'perfil', component: PerfilUsuarioComponent  },
  { path: 'admin/usuarios', component: UsuariosComponent  },
  { path: 'admin/gestionCentros', component: GestionCentrosComponent  },
  { path: 'admin/crearUsuario', component: CrearUsuarioComponent  },
  { path: 'admin/tiposTaller', component: CrearTipoTallerComponent  },
  { path: 'organizador/crearTaller', component: CrearTallerComponent, canActivate:
[AuthGuard, AuthGuardOrganizador]  },
  { path: 'organizador/modificarTaller', component: ModificarTallerComponent,
canActivate: [AuthGuard, AuthGuardOrganizador]  },
  { path: 'organizador/eliminarTaller', component: EliminarTallerComponent,
canActivate: [AuthGuard, AuthGuardOrganizador]  },
  { path: 'organizador/verReservas', component: VerReservasComponent, canActivate:
[AuthGuard, AuthGuardOrganizador]  },
  { path: 'organizador/estadisticas', component: EstadisticasComponent, canActivate:
[AuthGuard, AuthGuardOrganizador]  },
  { path: 'reserva/:idTaller', component: ReservaNoRegistradoComponent  },
  { path: 'reservar/:idTaller', component: ReservaRegistradoComponent  },
  { path: 'reservas', component: VerReservasUsuarioComponent  },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
  providers: [AuthGuard, AuthGuardOrganizador, AuthGuardAdmin, AuthGuardVisitor]
})
export class AppRoutingModule {

}
```

Imagen 9. App-Routing Module

En este archivo, no solo se definen las rutas de los componentes, sino que también permite gestionar los roles de la aplicación. Es posible añadir unos archivos que son ejecutados antes de la carga del componente llamados Guardas. Gracias a estos archivos se puede comprobar el tipo de usuario y bloquear el acceso o permitirlo en función del rol que tenga en la aplicación. Para el correcto funcionamiento de esta he creado cinco guardas que gestionan los roles y el servicio de autenticación. Tres de ellos se encargan de la gestión de los roles; ***auth.guardAdmin.ts***, ***auth.guardOrganizador.ts***

y ***auth.guardVisitor.ts***. Estos archivos se encargan de recoger un parámetro almacenado en almacenamiento interno del navegador que indica los permisos del usuario en la aplicación. Este parámetro (enviado por el servidor) se introduce en el momento en el que el usuario se autentica en la plataforma. Posteriormente, se realiza una comprobación contra el servidor y si la comprobación de la guarda resulta positiva, esta permitirá la ejecución del componente. En caso contrario, provocará la redirección del usuario a la página de *Login*. En la siguiente captura se muestra un ejemplo del funcionamiento de la guarda del administrador:

```
@Injectable()
export class AuthGuardAdmin implements CanActivate {
    constructor(private loginService: LoginService, private router:
Router){}
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) :
boolean | Observable<boolean> | Promise<boolean> {
        const role = localStorage.getItem('role');
        if(role!="admin")
        {
          this.router.navigate(['/login']);
          return false;
        }
        else
        {
          const userId = localStorage.getItem('userId');
          return this.loginService.checkRole(userId, role).then(respuesta =>
{            if(respuesta['check']){return true;}
            else{this.router.navigate(['/login']);
              return false;
            }
```

Imagen 10. Guarda de administrador

El resto de archivos se encargan de la autenticación del usuario en la plataforma; ***auth.guard.ts*** y **auth-interceptor.ts**. El primer archivo se encarga de comprobar que el usuario esta autenticado en la aplicación, sin importar el rol de usuario:

```
@Injectable()
export class AuthGuard implements CanActivate {
    constructor(private loginService: LoginService, private router:
Router){}
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot)
: boolean | Observable<boolean> | Promise<boolean> {
        const isAuth = this.loginService.getIsAuth();
        if(!isAuth)
        {
          this.router.navigate(['/login']);
        }
        return isAuth;
    }
}
```

Imagen 11. AuthGuard

Se realiza una llamada a un método del loginService (un servicio del que hablaré a continuación) que verifica la autenticación. Devuelve el valor de una variable booleana que indica si el usuario se ha autenticado o no.

1.1.3 Servicios:

¿Pero qué es un servicio? Un servicio en Angular es un archivo de tipo *typescript* que sirve para compartir funcionalidad entre componentes. El patrón de diseño que se utiliza en este tipo de archivos es el ***Singleton***. Un patrón *Singleton* actúa como un recurso compartido donde se define una funcionalidad global y que puede ser referenciado desde cualquier lugar de la aplicación para poder utilizar sus funciones.

En la construcción he definido dos servicios; el *loginService* del que he hablado un poco anteriormente y el *petitionService*.

El ***loginService*** alberga todas las funciones y todos los parámetros necesarios para el funcionamiento de la autenticación de la aplicación. Los métodos más importantes de este servicio son aquellos que se encargan de la creación de los usuarios y los que realizan el proceso de autenticación.

En la siguiente imagen se muestra uno de los métodos encargados de la creación de usuarios. Este en concreto es el que se utiliza desde la pantalla de registro. Recibe como parámetros todos los datos introducidos por el usuario y realiza una petición POST a la API REST para la creación del usuario enviando los anteriores datos en el cuerpo de la petición.

```
createUser(email: string, password: string, centro: string,
fechaNacimiento: Date, nombre: string, telefono: string)
  {
    this.http.post("http://localhost:3000/api/users/signup",{
      "email": email,
      "password": password,
      "centro": centro,
      "nombre": nombre,
      "telefono": telefono,
      "tipo": "none",
    }).subscribe(response => {
      console.log(response);
      window.location.reload();

    })
  }
```

Imagen 12. Método de creación de usuario

Para la autenticación del usuario todo comienza en el componente *Login* haciendo una llamada al siguiente método:

```
login(email: string, password: string)
  {
    return new Promise((resolve,reject) => {
      this.http.post<{token : string,  expiresIn: number, role:string,
userId: string, tipo:string}>("http://localhost:3000/api/users/login", {
      email: email,
      password: password,
    }).subscribe( response => {
      const token = response.token;
      const role = response.role;
      const userId = response.userId;
      const tipo = response.tipo;
      this.token = token;
      if (token)
      {
          const expiresInDuration = response.expiresIn;
          this.setAuthTimer(expiresInDuration);
          this.isAuthenticated = true;
          this.authStatusListener.next(true);
          const now = new Date();
          const expirationDate = new Date(now.getTime() + expiresInDuration
* 1000);
          console.log(expirationDate);
          this.saveAuthData(token, expirationDate, role, userId, tipo);
          this.router.navigate(["/home"]);
      }
      resolve(response);
    });
```

Imagen 13. Método Login

Este método realiza una petición POST a la API enviando el email y la contraseña introducida por el usuario. El servidor verifica si el email coincide con la contraseña introducida y en caso afirmativo el servidor devuelve un token y su tiempo de expiración, un rol, el identificador del usuario y el tipo de actividad que realiza (en el caso de los organizadores). Se comprueba si el token recibido tiene contenido y en caso positivo se asignan los valores recibidos a las variables globales del servicio. También se ejecuta el siguiente método que se encarga de guardar los valores necesarios en el almacenamiento local del navegador para mantenerlos disponibles durante todo el ciclo de vida de la aplicación ya que cada vez que se cambia de componente se borran todos los datos de los servicios.

```
private saveAuthData(token: string, expirationDate: Date, role: string,
userId: string, tipo:string) {
  this.role=role;
  localStorage.setItem("token", token);
  localStorage.setItem("expiration", expirationDate.toISOString());
  localStorage.setItem("role", role);
  localStorage.setItem("userId",userId);
  localStorage.setItem("tipo",tipo);
}
```

Imagen 14. Información de usuario. Local Storage

Cuando un usuario cierra sesión, se ejecuta un método que realiza el proceso opuesto, se borran todos esos datos del navegador:

```
private clearAuthData() {
  this.role=null;
  localStorage.removeItem("token");
  localStorage.removeItem("expiration");
  localStorage.removeItem("role");
  localStorage.removeItem("userId");
  localStorage.removeItem("tipo");
}
```

Imagen 15. Borrado del local storage

1.1.4 Construcción del calendario:

Para la construcción del calendario he decidido utilizar una librería de código abierto alojada en GitHub que proporciona un calendario realizado mediante JavaScript de forma gratuita. Esta librería se llama ***FullCalendar***. Es posible adquirir una opción premium de la librería, la cual no necesitaba ya que la versión gratuita me proporcionaba todas las herramientas necesarias para lo que previamente se había planeado.

Esta librería permite añadir eventos en el calendario y, además, es posible visualizarlos en distintos formatos (por mes, semana, día o listado). *FullCalendar* se integra perfectamente con Angular debido a que la librería proporciona un componente adaptado a la API estándar de *fullCalendar*.

Para poder hacer uso de este componente, es necesario instalarlo previamente en nuestro proyecto mediante el siguiente comando:

```
npm install --save @fullcalendar/angular @fullcalendar/core @fullcalendar/daygrid
```

Posteriormente, hay que importar el módulo de *FullCalendar* en el módulo raíz de la aplicación y ya estaría listo para usarse. Tras esto, he creado dos componentes para la creación del calendario; uno para el calendario público y otro para el calendario interno de la aplicación. El grueso del código para la generación del calendario se encuentra en el archivo *typescript*. En las siguientes imágenes se muestra cómo se genera una instancia del calendario y cómo se cargan los talleres almacenados en la base de datos.

```
import dayGridWeek from '@fullcalendar/daygrid';
import dayGridDay from '@fullcalendar/daygrid';
import { Taller } from '../model';
import esLocale from '@fullcalendar/core/locales/es';

@Component({
  selector: 'app-calendario',
  templateUrl: './calendario.component.html',
  styleUrls: ['./calendario.component.css'],
  providers: [ DatePipe],
})
export class CalendarioComponent implements OnInit {

  calendarPlugins = [dayGridPlugin, timeGridPlugin, timeGridWeek,
timeGridDay, dayGridDay, dayGridWeek, dayGridMonth];
  talleres = [];
  calendar: Calendar;
  mapaColores: Map<string, string> = new Map<string, string>();

  @ViewChild('calendar') calendarComponent: FullCalendarComponent;

  constructor(private petitionService: PetitionService, private datePipe:
DatePipe, public dialog: MatDialog, private router: Router) { }

  ngOnInit() {

    let newEvents=[];
    let calendarEl: HTMLElement = document.getElementById('myCalendar')!;
    this.calendar = new Calendar(calendarEl,
      {
      locale: 'es',
      plugins: [ dayGridPlugin, timeGridPlugin, timeGridWeek, timeGridDay,
dayGridDay, dayGridWeek, dayGridMonth ],
      header: {
        left: 'prev,next today',
        center: 'title',
        right: 'dayGridMonth,timeGridWeek,timeGridDay,listWeek'
      },
eventClick: function(info) {
```

Imagen 16. Código calendario (1/2)

```
,eventClick: function(info) {
        console.log(info);
        info.el.style.borderColor = 'red';
      }

    });
```

```
    this.petitionService.getAllTipos().then(listadoTipos => {
      listadoTipos['tipos'].forEach(element => {
        this.mapaColores.set(element['nombre'],element['color']);
      });

      this.petitionService.getAllTalleres().then(listadoTalleres => {
        listadoTalleres['talleres'].forEach((taller: Taller) => {
          const fecha = new Date(taller.fecha);
          let hora1 : number = Number.parseInt(taller.hora.split(':')[0]);
          let hora2 = taller.hora.split(':')[1];
          hora1 = hora1 + taller.duracion;

          const startDate = this.datePipe.transform(new Date(taller.fecha),
'yyyy-MM-dd')+'T'+taller.hora+':00';
          const endDate = this.datePipe.transform(new Date(taller.fecha),
'yyyy-MM-dd')+'T'+hora1+':'+hora2+':00';

          this.calendar.addEvent({ title: taller.nombre+'
('+taller.plazasLibres+' plazas)', color: this.mapaColores.get(taller.tipo),
start: startDate, duration:{hours:4}, allDay: false, url:
'http://localhost:4200/reservar/'+taller['_id']});
          newEvents.push({title:taller.nombre,
date:this.datePipe.transform(new Date(taller.fecha), 'yyyy-MM-dd')});
        });

      });
    });
    this.calendar.addEventSource(newEvents);
    this.calendar.render();

  }
```

Imagen 17. Código calendario (2/2)

2. CONSTRUCCIÓN DEL BACK-END:

Para la construcción del servidor he utilizado *NodeJS* que es un entorno *JavaScript* que está diseñado para construir aplicaciones web con un alto nivel de optimización. Está basado en eventos utilizando el motor V8 desarrollado por Google. Gracias al aprovechamiento de ese motor, *Node* proporciona un entorno de ejecución del lado del servidor que compila y ejecuta javascript a grandes velocidades. Esto se debe a que la compilación se realiza en código de máquina nativo.

Node fue diseñado para generar un sistema escalable que fuese capaz de soportar un alto número de conexiones de forma simultánea. Este problema común en las plataformas web, ha sido pensado mediante un bucle de eventos de tipo asíncrono. Gracias a esto, las peticiones que se hacen a la API se tratan como eventos y todas pertenecen a un único bucle. *NodeJS* emplea la asincronía para realizar la ejecución de tareas de forma paralela lo que permite tratar varias peticiones de forma simultánea evitando los bloqueos en el flujo de trabajo.

Para utilizar *NodeJS* es necesario tenerlo instalado previamente en nuestro dispositivo. Es posible instalarlo mediante un instalador proporcionado en la página oficial o mediante un conjunto de comandos.

Nuestro *backend* nos permitirá conectar el *front* de la aplicación con nuestra base de datos mediante el uso de peticiones para recuperar, modificar o añadir datos en ella. Para ello, he hecho una API REST, es decir, un servidor que tiene definidas rutas para ejecutar diferentes códigos en función de la ruta elegida. De esta forma, el cliente de angular puede hacer solicitudes para enviar o recibir datos al servidor.

La construcción del servidor ha sido desarrollada en otro proyecto con lo que el proyecto Angular está totalmente separado del *back-end*. Para ello se ha utilizado también **Express**, un *framework* para nodeJS que permite que el desarrollo sea mucho más sencillo. Para utilizarlo tan solo hay que instalar su librería en el proyecto del servidor.

En ese proyecto he creado un archivo llamado ***server.js*** que se encarga de iniciar el servidor. En él, se definen las librerías necesarias para el funcionamiento del servidor como por ejemplo HTTP o CORS. También se define el puerto en el que el servidor se levanta y se define un método manejador de los errores de las peticiones. Hay también un apartado para definir las cabeceras permitidas en las peticiones y finalmente se levanta el servidor:

```
const app = require("./backend/app");
const debug = require("debug")("node-angular");
const http = require("http");
const cors = require('cors');

const normalizePort = val => {
  var port = parseInt(val, 10);
  if (isNaN(port)) {
    return val;
  }

  if (port >= 0) {
    return port;
  }

  return false;
};

const onError = error => {
  if (error.syscall !== "listen") {
```

Imagen 18. Server.js (1/2)

```
if (error.syscall !== "listen") {
    throw error;
  }
  const bind = typeof port === "string" ? "pipe " + port : "port " + port;
  switch (error.code) {
    case "EACCES":
      console.error(bind + " requires elevated privileges");
      process.exit(1);
      break;
    case "EADDRINUSE":
      console.error(bind + " is already in use");
      process.exit(1);
      break;
    default:
      throw error;
  }
};

const onListening = () => {
  const addr = server.address();
  const bind = typeof port === "string" ? "pipe " + port : "port " + port;
  debug("Listening on " + bind);
};

const port = normalizePort(process.env.PORT || "3000");
app.set("port", port);
```

```
app.use(cors());
app.use((req, res, next) => {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Methods", "GET, PUT, POST, DELETE");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
Content-Type, Accept, Authorization");
  next();
  });

const server = http.createServer(app);
server.on("error", onError);
server.on("listening", onListening);
server.listen(port);
```

Imagen 19. Server.js (2/2)

Como podemos ver en la imagen anterior, hay un archivo llamado ***app.js***. Este archivo es el encargado de albergar la aplicación express de nodeJS. Se encarga de importar todos los modelos que se usarán para la base de datos, realizar la conexión con la misma y definir todas las rutas que formarán la API REST con su respectivo código interno. Además, también se ocupa de realizar el volcado inicial de

datos en la base de datos, es decir, introduce los centros, los talleres y el usuario administrador que previamente han sido definidos en unos archivos JSON que se encuentran en un directorio de la aplicación llamado ***"deploy".***

```
const express = require('express');
const mongoose = require('mongoose');
const Taller = require('./models/model');
const Persona = require('./models/persona');
const Reserva = require('./models/reserva');
const Solicitud = require('./models/solicitud');
const Centro = require('./models/centro');
const Tipo = require('./models/tipo');
const Recuperacion = require ('./models/recuperacionPassword');
const User = require('./models/user');
const cors = require('cors');
const userRoutes = require("./user");
const checkAuth = require("./middleware/check-auth");
const uuidv1 = require('uuid/v1');
const bcrypt = require("bcrypt");
const app = express();

mongoose.connect("mongodb://localhost:27017/tfg").then(console.log("Connected
to BD")).catch(error => console.log(error));
mongoose.set('debug', true);

const fs = require('fs');
let rawdata = fs.readFileSync('./deploy/centros.json');
let centros = JSON.parse(rawdata);

let rawdata2 = fs.readFileSync('./deploy/actividades.json');
let actividades = JSON.parse(rawdata2);

let rawdata3 = fs.readFileSync('./deploy/user.json');
let users = JSON.parse(rawdata3);

var bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(cors());

for(var i=0;i<centros['centros'].length;i++)
{
  console.log(centros['centros'][i]);
  Centro.findOneAndUpdate(
    {nombre:centros['centros'][i]['nombre'], localidad:
```

Imagen 20. App.js (1/2)

```
  Centro.findOneAndUpdate(
    {nombre:centros['centros'][i]['nombre'], localidad:
centros['centros'][i]['localidad']},
    {nombre:centros['centros'][i]['nombre'], localidad:
centros['centros'][i]['localidad']},
    {upsert:true}).then(result => {
  });
}

for(var i=0;i<actividades['actividades'].length;i++)
{
  Tipo.findOneAndUpdate(
    {nombre:actividades['actividades'][i]['nombre']},
    {nombre:actividades['actividades'][i]['nombre'],
informacion:actividades['actividades'][i]['informacion'],lugar:actividades
['actividades'][i]['lugar'],
color:actividades['actividades'][i]['color']},
    {upsert:true}
    )
}

console.log(users);
bcrypt.hash(users['user']['password'],10).then(hash => {
  User.findOneAndUpdate(
    {nombre:users['user']['nombre']},

{email:users['user']['email'],notificaciones:users['user']['notificaciones
'], role:users['user']['role'],
centro:users['user']['centro'],password:hash},
    {upsert:true}
    );
});
```

```
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept, Authorization"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PATCH, PUT, DELETE, OPTIONS"
  );
  next();
});
```

Imagen 21. App.js (2/2)

Como acabo de mencionar, en este archivo se declaran todas las rutas y funcionalidades del servidor. Es decir, se pueden definir entre otras, peticiones POST, GET, DELETE… En la siguiente imagen se ve un ejemplo de cómo funciona:

```
app.post('/api/talleres',checkAuth,(req,res,next) => {
    console.log("adding taller");
    console.log(req.body.fecha);
    const taller = new Taller({
      nombre: req.body.nombre,
      tipo: req.body.tipo,
      fecha: req.body.fecha,
      hora: req.body.hora,
      plazasLibres: req.body.plazas,
      plazasOcupadas: 0,
      duracion: req.body.duracion,
      organizador: req.body.organizador

    })

    taller.save().then(tallerAdded => {
      res.status(201).json({
        message: "Taller added successfully",
        taller: taller
      });
    });
    console.log(taller);

});
```

Imagen 22. Ejemplo endpoint

Se define la ruta **"/api/talleres"** para las peticiones POST. Este método en concreto servirá para crear un taller de una actividad. Como podemos ver, se verifica la autenticación del usuario y tras esto se recogen los parámetros del cuerpo de la petición para crear un objeto de tipo taller y guardarlo en la BD. Una vez que el proceso se ha realizado, se devuelve una respuesta al lado del cliente.

Para gestionar la seguridad del servidor he tenido que desarrollar un servicio de autenticación de usuarios. De esta forma, es posible añadir a los métodos un parámetro que indica si hay que realizar o no la verificación. Todos aquellos que lleven el parámetro ***checkAuth*** no permitirán que alguien extraño al sistema ejecute dichas rutas del servidor.

Esa funcionalidad esta implementada en un archivo llamado ***check-auth.js*** y lo que hace es comparar el token del usuario enviado desde el lado del cliente con el secreto con el que se generan esos tokens. De esa forma se puede saber si el token es de la aplicación o no, autorizando o denegando el acceso a las rutas.

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1]
    jwt.verify(token, "secret_of_EXPERIMENTA_application_2019");
    next();
  }catch(error){
    res.status(401).json({message: "Auth failed"});
  }
}
```

Imagen 23. Check-auth

Para la generación y verificación de dichos tokens he utilizado una librería que se llama ***jsonwebtoken***. Es el mismo token del que he hablado previamente en la gestión de usuario de la parte del frontal. Cuando un usuario introduce sus credenciales, se hace una llamada post a la ruta *"/login"* si se encuentra un usuario con ese correo electrónico, entonces se encripta la clave recibida y se compara con la almacenada en la base datos. Si estas coinciden se crea un token a partir del secreto de la aplicación, el correo y el identificador del usuario, indicando también el tiempo de expiración. Finalmente se emite una respuesta que engloba ese token junto con otros datos del usuario que la aplicación necesitará.

```
router.post("/login", (req, res,next)=>{
    let fetchedUser;
    User.findOne({email:req.body.email}).then(user => {
      if(!user)
      {
        return res.status(401).json({
          message: "User not exists"})
      }
      fetchedUser=user;
      return bcrypt.compare(req.body.password,user.password)
    })
    .then(result => {
      if(!result){
        return res.status(401).json({
          message: "Auth failed"});
      }

      const token = jwt.sign({email: fetchedUser.email, userId:
fetchedUser._id}, 'secret_of_EXPERIMENTA_application_2019', {expiresIn:
'1h'});
      res.status(200).json({
        token: token,
        expiresIn: 3600,
        role: fetchedUser.role,
        userId: fetchedUser._id,
        tipo: fetchedUser.tipo,
```

Imagen 24. Login endpoint (1/2)

```
tipo: fetchedUser.tipo,
      })
    })
    .catch(err => {
      return res.status(401).json({
        message: "Auth failed"
      });
    })
});
module.exports = router;
```

Imagen 25. Login Endpoint (2/2)

3. Construcción de la base de datos:

Para el almacenamiento de datos de la aplicación he decidido utilizar **MongoDB**. MongoDB es una base de datos NoSQL que almacena documentos y colecciones. Los documentos son archivos similares a JSON que corresponderían a los típicos esquemas de los lenguajes relacionales.

He elegido esta base de datos ya que es muy rápida y es flexible. MongoDB permite adaptar la base de datos a los cambios que vayan surgiendo durante el proceso de desarrollo. Esta fue una característica que valoré bastante a la hora de elegir esta herramienta ya que había algunos campos de las tablas del diseño de la BD de los que no estaba totalmente convencido.

MongoDB ofrece dos opciones, crear tu propia base de datos en tu dispositivo o utilizar el almacenamiento cloud que ofrecen. Para el proyecto he decidido crear mi propia base de datos que estará alojada en un servidor de la universidad.

Para poder construir una base de datos con mongoDB es necesario instalarlo en nuestro dispositivo. Se puede hacer mediante comandos (como veremos en el anexo de configuración del servidor) o mediante los sencillos instaladores ofrecidos en la página oficial. Una vez instalado se crea una conexión a la BD configurando el puerto, el nombre del host, el método de autenticación y las propiedades para el protocolo SSL[5].

Como herramienta para manejar la BD he utilizado ***mongoDB Compass***, que permite ver todas las bases de datos, sus colecciones y los datos que guardan. Las colecciones se pueden crear desde esta herramienta o, si una colección no existe, esta se crea por defecto cuando se intenta guardar un documento que corresponde con el esquema definido para la colección.

Los modelos que definen estas colecciones se encuentran alojados en archivos del servidor. En la siguiente imagen veremos un ejemplo de cómo se definen los esquemas. Estos esquemas hay que importarlos en el archivo app.js para poder utilizarlos.

[5] SLL es el acrónimo de Secure Sockets Layer. Es la tecnología estándar para mantener conexiones seguras en internet y proteger la información de una comunicación entre dos sistemas.

```
const mongoose = require('mongoose');
const uniqueValidator = require("mongoose-unique-validator");
```

```
// MODELO BD PERSONA:
const userSchema = mongoose.Schema({
  nombre: { type: String},
  email: {type: String, required: true, unique:true},
  password: {type: String},
  telefono: {type: String, required: true},
  centro: {type: String, required: true},
  fechaNacimiento: { type: Date},
  role: {type: String, required: true},
  tipo: {type: String},
  notificaciones: {type: Boolean, required:true}
});

userSchema.plugin(uniqueValidator);
module.exports = mongoose.model('User', userSchema);
```

Imagen 26. Ejemplo de esquema

En estos archivos se definen todos los campos que tiene que tener el documento, indicando su tipo, su unicidad y si son campos obligatorios o no. Finalmente, el schema se exporta para que pueda ser importado desde otros archivos del servidor.

4. Librerías usadas:

Durante la construcción del proyecto he utilizado varias librerías que me han ayudado a construir una aplicación más bonita y de forma más rápida y sencilla. Las librerías utilizadas son las siguientes:

- **EmailJS**. Para el envío de correos electrónicos desde el servidor. (https://www.npmjs.com/package/emailjs)
- **UUID**. Para generar códigos de identificación únicos a la hora de solicitar un cambio de contraseña. (https://www.npmjs.com/package/uuid).
- **Ng2-charts**. Para la creación de gráficos en el apartado de estadísticas. (https://valor-software.com/ng2-charts/#GeneralInfo).
- **FullCalendar**. Para la construcción del calendario en el que se ven los talleres. (https://fullcalendar.io/)
- **Angular CLI**. Para la automatización de aplicaciones Angular.
- **Angular Material**. Para usar los elementos predefinidos disponibles en la página oficial de Angular.

CAPÍTULO 5. TEST Y PRUEBAS

En este capítulo trataré las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación y las evaluaciones que he realizado con distintos perfiles de personas para comprobar su usabilidad.

5.1 PRUEBAS

Durante la planificación del proyecto se estableció un listado de pruebas para realizar al finalizar el desarrollo de la aplicación. Las pruebas realizadas son las siguientes:

- **Login de usuario.** Se ha comprobado que se puede entrar a la aplicación correctamente con todos los tipos de usuarios registrados. Además, también se ha comprobado que no se puede entrar introduciendo datos incorrectos ni introduciendo enlaces en la URL.
- **Alta de taller.** Se ha realizado la creación de un taller con el usuario de organizador. Se ha verificado el contenido de la base de datos para comprobar que todos los datos introducidos correspondían con lo que se había creado. Además, también se ha comprobado que el taller creado aparece en el calendario en el día elegido para ello.
- **Baja de taller.** Se ha comprobado que al eliminar un taller desaparece su registro de la base de datos y también deja de ser visible desde el calendario de talleres. La comprobación se ha realizado con el usuario organizador y administrador.
- **Modificar taller.** Se ha realizado una edición con el usuario organizador. Al hacer efectivos los cambios se ha comprobado que se ha cambiado el contenido en la base de datos.
- **Creación de reserva.** Esta funcionalidad se ha comprobado con varios usuarios, tanto con el usuario no registrado como con el usuario registrado Solicitante. En el caso del usuario no registrado, se ha accedido al calendario mediante la pantalla de inicio de la app, se ha seleccionado un taller y se ha realizado la reserva. Se ha comprobado que el registro aparecía en la base de datos, que se han restado las plazas reservadas del taller seleccionado y también que ha llegado la confirmación de reserva tanto al email del usuario como al email del organizador.En el caso del Solicitante se ha realizado la misma comprobación, pero desde el calendario interno de la aplicación.
- **Solicitud de eliminación de reserva.** Se ha comprobado con el usuario Solicitante a enviar una solicitud de eliminación de reserva. Se ha comprobado que le llega al organizador correspondiente un correo de la solicitud.
- **Solicitud de modificación de reserva.** Se ha comprobado con el usuario Solicitante a enviar una solicitud de modificación de reserva. Se ha comprobado que le llega al organizador correspondiente un correo de la solicitud.
- **Gestión de centros.** Se ha comprobado con el Administrador que pueda crear, editar y modificar centros. Además, se ha comprobado que estos cambios se quedan guardados en la base de datos y se ven reflejados posteriormente en la aplicación.
- **Cierre de sesión.** Se ha comprobado que después de cerrar sesión con cualquier usuario registrado no se puede volver a entrar. Ni haciendo clic en volver atrás ni mediante un enlace conocido.
- **Creación de usuario.** Con el usuario administrador se han creado varios usuarios con diferentes roles. Después de crearlos se ha comprobado que todo estaba guardado en la base de datos y se ha cerrado sesión para entrar de nuevo a la aplicación con cada uno de esos usuarios. Para cada usuario con un rol distinto se ha comprobado que tenía los permisos adecuados.

- **Eliminación de una reserva.** Se ha procedido a la eliminación de una reserva con el Organizador. Se ha comprobado que los cambios se hacen efectivos en la base de datos y que se restauran las plazas de la reserva en el taller correspondiente.
- **Inserción de logos.** Se ha procedido a la inserción de varios logos con el perfil del administrador y después se ha comprobado que estos logos aparecían en la página de inicio de la aplicación.
- **Asignación de colores.** Se han creado varias actividades con el perfil del administrador asignando colores distintos. Se ha comprobado que los colores se almacenaban en la base de datos con el formato correcto y que los talleres creados para ese tipo de actividad aparecían pintados en el calendario con el color indicado.
- **Gestión de actividades.** Se ha procedido a la creación, modificación y eliminación de una actividad y se ha comprobado después de cada funcionalidad el estado de los datos en la BD.
- **Cambiar contraseña**. Se ha comprobado con todos los usuarios registrados que el cambio de contraseña que se realiza desde el perfil del usuario se hace efectivo. Se ha comprobado haciendo *logout* y volviendo a iniciar sesión con la nueva contraseña.

5.2 TEST

La evaluación de la usabilidad implica analizar el entorno y los usuarios que van a utilizar el producto. Por ello, he decidido buscar diferentes perfiles que podrían encajar con los usuarios reales que tendrá la aplicación. El desarrollo de sistemas interactivos centrados en el usuario evaluando la usabilidad nos permitirá desarrollar un producto que produzca una mayor satisfacción al usuario. Además, esto permitirá evitar rediseños en fases posteriores.

El método que he decidido utilizar para realizar las evaluaciones es un método de tipo Test, concretamente el método "***Thinking Aloud***" (Pensando en voz alta) mezclado con una tabla heurística (para usuarios expertos). Mediante este método de evaluación, se les pide a los usuarios que expresen en voz alta sus sentimientos y opiniones mientras que interaccionan con el sistema.

El principal beneficio de este método es mejorar la comprensión del modelo mental del usuario y la interacción con el usuario. No obstante, hay otros beneficios como conocer la terminología que utiliza un usuario para expresar una idea (en nuestro caso talleres, actividades, reservas…). Este método se puede realizar en todas las fases del ciclo de vida de un producto, aunque yo lo he utilizado casi en fase de preproducción.

Sin embargo, no tendremos que tener en cuenta todas las opiniones de los usuarios ya que la información proporcionada será subjetiva y selectiva, lo que nos obligará a filtrar los cambios y mejoras sugeridas. Para el procedimiento es necesario proporcionar a los usuarios el producto que hay que probar y un conjunto de tareas a realizar. Para recoger el resultado de este proceso he desarrollado una plantilla en la que apuntaré los datos del usuario, la dificultad que ha tenido cada tarea para el usuario y todas las opiniones que ha diseño durante la evaluación. Estas plantillas pueden verse en el anexo III.

Tras realizar varias evaluaciones he podido comprobar que había partes de la aplicación que no se entendían bien o resultaban algo confusas. Esto me ha permitido hacer pequeños cambios en el diseño para poder hacerlo más usable y cercano al tipo de público al que se orientará la aplicación. Además, durante el desarrollo de las evaluaciones surgieron errores o pequeños bugs que no había detectado durante el desarrollo de la aplicación. Estas pruebas me han permitido corregir estos pequeños errores y mejorar la eficiencia de la aplicación para evitar algunos bugs que aparecieron por problemas de sincronización.

CAPÍTULO 6. GESTIÓN DEL PROYECTO.

1. ALCANCE

1.1 Requisitos:

La siguiente tabla incluye los requisitos inicialmente planeados para el desarrollo de la aplicación:

Código	Descripción	Fecha
R1	La aplicación tendrá un calendario que permita ver los talleres disponibles.	04/02/2019
R2	Será posible crear talleres desde la aplicación y solo podrán crearlos los administradores o los organizadores.	04/02/2019
R3	Será posible hacer reservas de los talleres visibles en el calendario.	04/02/2019
R4	Los administradores y organizadores podrán modificar o eliminar los talleres	04/02/2019
R5	Los administradores y organizadores podrán modificar o eliminares reservas.	04/02/2019
R6	El administrador podrá crear usuarios.	04/02/2019
R7	El administrador podrá crear actividades.	04/02/2019
R8	Los administradores y los organizadores tendrán disponible un apartado de estadísticas para ver el estado de los talleres.	04/02/2019
R9	Habrá dos tipos de usuarios visitantes, el solicitante y el no registrado.	04/02/2019
R10	Sistema de roles que se active después de autenticarse.	04/02/2019
R11	Los usuarios registrados podrán solicitar la modificación o la eliminación de una reserva.	04/02/2019
R12	El administrador podrá cambiar los permisos de un usuario.	04/02/2019
R13	Todos los usuarios podrán modificar la información almacenada de su usuario.	04/02/2019
R14	Los usuarios podrán recuperar su contraseña en caso de no acordase de ella o perderla.	04/02/2019
R15	Los usuarios podrán cambiar su contraseña desde el interior de su cuenta.	04/02/2019

Tabla 8. Recopilación inicial de requisitos

1.2 Incidencias y cambios:

Se han añadido nuevos requisitos respecto al plan inicial con la finalidad de enriquecer la funcionalidad global de la aplicación.

Código	Descripción	Fecha
R16	Opción para activar o desactivar notificaciones de emails desde el perfil de usuario.	14/05/2019
R17	Sistema automatizado para insertar logos en la página principal	14/05/2019
R18	Creación de un sistema para personalizar la imagen del home de la aplicación.	14/05/2019
R19	Sistema que permita enviar mensajes entre los administradores y los organizadores.	14/05/2019

Tabla 9. Nuevos requisitos

2. TIEMPO

2.1 Cronograma real:

Las siguientes figuras contienen el diagrama Gantt del proyecto. El color azul representa los periodos planificados para cada parte del proyecto. El color verde corresponde con el periodo real.

	Febrero 2019				Marzo			
	4	11	18	25	4	11	18	25
DOP								
Análisis								
Diseño								
Implementación								
Pruebas								
Defensa								
Control y seguimiento								

Tabla 10. Cronograma real febrero y marzo

	Abril 2019					Mayo 2019			
	1	8	15	22	29	6	13	20	27
DOP									
Análisis									
Diseño									
Implementación									
Pruebas									
Defensa									
Control y seguimiento									

Tabla 11. Cronograma real abril y mayo

	Junio 2019			
	3	10	17	24
DOP				
Análisis				
Diseño				
Implementación				
Pruebas				
Defensa				
Control y seguimiento				

Tabla 12. Cronograma real junio

2.2 Desviaciones:

Como se puede apreciar en las figuras anteriores hay algunas partes del proyecto en las que los periodos planificados no concuerdan con el periodo real en el que se ha realizado:

- **D1-CS**. El primer desvío se ha producido en la realización del control y seguimiento ya que cuando comencé a hacer la recogida de requisitos no tenía preparado un archivo de seguimiento. No afectó a otras partes del proyecto, pero hubo una mala gestión durante las dos primeras semanas.
- **D2-IMP**. La implementación comenzó dos semanas más tarde de lo previsto por motivos personales. Es una desviación con importancia que se ha solucionado añadiendo horas de trabajo durante toda la semana para recuperar las perdidas.
- **D3-IMP**. La implementación se ha tenido que prolongar una semana más de lo esperado para solucionar problemas encontrados durante las evaluaciones y para terminar otras sugerencias de mejoras aportadas por el tutor.
- **D4-PRU**. Las pruebas tuvieron que prolongarse también una semana para comprobar las últimas implementaciones realizadas.
- **D5-DEF**. La preparación de la defensa (ensayos, presentación *power point* y cartel) se había planificado para todo el mes de junio. Sin embargo, todas las horas necesarias para ello se concentrarán en la última semana de junio y la primera de julio por razones laborales y debido a otros desvíos comentados previamente.

2.3 Estimaciones de dedicación:

En esta tabla se muestran las estimaciones de horas que se habían realizado al comienzo del proyecto y como se han ido cumpliendo a final de cada mes.

Paquete de trabajo	Horas						
	Plan	**Hasta** 28/02	**Hasta** 31/03	**Hasta** 30/04	**Hasta** 31/05	**Hasta** 24/06	**%Des**
DOP	12	12	12	12	12	12	0%
Análisis	5	6	6	6	7	7	-40%
Diseño	10	7	7	7	7	7	+30%
Implementación	345	0	27	135	243	350	-2%
Pruebas	3	0	0	0	1	3	0%
Defensa	2	0	0	0	0	0	0%
Control y seguimiento	8	1	3	5	7	8	0%
Proyecto	385	27	56	166	278	388	-12%

Al final del proyecto se puede apreciar que ha habido una desviación total negativa del 12% en la gestión de las horas del proyecto. El mayor desfase se ha producido en la fase de implementación y en la fase del análisis de la aplicación. Sin embargo, hay que destacar que ha habido un desvío positivo del 30% en la fase de diseño de la aplicación.

3. COMUNICACIONES

Durante el proyecto se han realizado reuniones con el tutor para hacer un control del proyecto y sugerir mejoras e ideas para el sistema. Se pueden ver las actas de esas reuniones en el Anexo 2 del proyecto.

CAPÍTULO 7. CONCLUSIONES

El resultado global del proyecto ha sido satisfactorio ya que he conseguido desarrollar la aplicación con todos los requisitos establecidos por el cliente (en mi caso Eloy Mata) sin grandes desviaciones de tiempo, aunque haya sido necesario aportar más tiempo del planeado al proyecto.

Desde el punto de vista personal, me ha supuesto un gran reto, ya que durante este semestre he estado en Madrid trabajando y terminando las asignaturas que me quedaban a distancia. He tenido que distribuir bien las horas libres para poder sacar el proyecto adelante. Además, he desarrollado el proyecto con tecnologías que no he visto durante la carrera y de las que he ido aprendiendo en mi trabajo de Madrid. La mayor parte de las cosas las he tenido que hacer de forma autodidacta y consultando tutoriales e información en Internet.

En ocasiones he estado atascado debido a los pocos conocimientos que tenía de estas nuevas tecnologías, pero poco a poco lo he ido desarrollando todo y me ha servido para aprender muchísimo sobre ellas. En general, podría decir que gracias al desarrollo del TFG me muevo de forma muy ágil con las tecnologías Angular, NodeJS y MongoDB.

Si hablamos de complejidad, podría decir que la parte que me ha resultado más tediosa ha sido el despliegue de la aplicación en el servidor de la universidad ya que era algo que no había hecho nunca y no soy un especialista en sistemas informáticos.

1. APRENDIZAJE PERSONAL:

Durante este proyecto se han aprendido conocimientos técnicos que enumero a continuación:

- Utilización del framework Angular 6 y su correspondiente lenguaje de programación Typescript.
- Utilización del framework NodeJS para el desarrollo de una API REST.
- MongoDB como base de datos para la gestión del almacenamiento del proyecto. Consultas y herramientas como Compass para gestionar la base de datos.
- Despliegue de un servidor de aplicaciones Apache.
- Despliegue de una aplicación Angular.
- Despliegue de una API Rest.
- Configuración de un servidor para un despliegue.
- Conocimiento de herramientas para el prototipado.
- Conocimiento de librerías usadas durante todo el desarrollo del proyecto.

2. MEJORAS:

Una vez finalizado el proyecto se exponen las posibles mejoras a realizar que mejorarían o ampliarían la funcionalidad del proyecto:

- Bloqueo de usuarios desde el perfil de administrador.
- Añadir índices a la base de datos para mejorar la eficiencia.
- Efectos de transición cuando cambia la pantalla.
- Añadir el sistema de CUASI a la aplicación para poder entrar mediante el servicio de gestión de usuarios de la universidad.

BIBLIOGRAFÍA:

1. Desarrollo en cascada. https://es.wikipedia.org/wiki/Desarrollo_en_cascada
2. MongoDB:
    a. https://es.wikipedia.org/wiki/MongoDB
    b. https://tecadmin.net/install-mongodb-on-centos/
3. NodeJS:
    a. https://nodejs.org/en/about/
    b. https://www.netconsulting.es/blog/nodejs/
4. Angular:
    a. https://angular.io/
    b. https://es.wikipedia.org/wiki/Angular_(framework)
    c. http://devopspy.com/linux/install-node-js-npm-angular-on-centos-7-x/
    d. https://blog.ng-classroom.com/blog/angular/Angular-Crear-Servicio/

5. UUID. https://es.wikipedia.org/wiki/Identificador_%C3%BAnico_universal
6. FullCalendar. https://fullcalendar.io/docs/angular
7. Jesús Lorés, Montse Sendín y Jordi Agost, Universitat de Lleida. **Evaluación**
8. PM2. http://pm2.keymetrics.io/docs/usage/quick-start/
9. EmailJS. https://www.npmjs.com/package/emailjs
10. Angular Charts. https://valor-software.com/ng2-charts/#GeneralInfo